AX5488
GPIB Interface Card

User's Manual

Disclaimers

The information in this manual has been carefully checked and is believed to be accurate. AXIOM Technology Co., Ltd. assumes no responsibility for any infringements of patents or other rights of third parties which may result from its use.

AXIOM Technology assumes no responsibility for any inaccuracies that may be contained in this document. AXIOM Technology makes no commitment to update or to keep current the information contained in this manual.

AXIOM Technology reserves the right to make improvements to this document and/or product at any time and without notice.

No part of this document may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of AXIOM Technology Co., Ltd.

©Copyright 1994 by AXIOM Technology Co., Ltd.

All rights reserved.

May 1994, 3rd Edition

Printed in Taiwan

Trademarks Acknowledgments

- AXIOM is a trademark of AXIOM Technology Co., Ltd.
- IBM is a registered trademark of International Business Machines Corporation.
- MS-DOS, Microsoft C and QuickBasic are trademarks of Microsoft Corporation.
- TURBO C is a trademark of Borland Inc.
- BASIC is a trademark of Dartmouth College.
- Intel is a trademark of Intel Corporation.
- Other brand names and trademarks are the properties and registered brands of their respective owners.

ESD Precautions

Integrated circuits on computer boards are sensitive to static electricity. To avoid damaging chips from electrostatic discharge, observe the following precautions:

Do not remove boards or integrated circuits from their anti-static packaging until you are ready to install them.

Before handling a board or integrated circuit, touch an unpainted portion of the system unit chassis for a few seconds. This helps to discharge any static electricity on your body.

Wear a wrist-grounding strap, available from most electronic component stores, when handling boards and components.

Unpacking

The AX5488 is packed in an anti-static bag. This board has components that are easily damaged by static electricity. Do not remove the anti-static wrapping until proper precautions have been taken. Safety instructions in front of this User's Manual describe anti-static precautions and procedures.

After unpacking the board, place it on a raised surface and carefully inspect the board for any damage that might have occurred during shipment. Ground the board and exercise extreme care to prevent damage to the board from static electricity.

Integrated circuits will sometimes come out of their sockets during shipment. Examine all integrated circuits, particularly the BIOS, processor and keyboard controller chip to ensure that they are firmly seated.

The AX5488 GPIB Interface Card package includes the following:

- AX5488 Board
- AS59099 DAC Driver CD
- AX5488 User's Manual
- Warranty Card

Make sure that all of the items listed above are present.

What To Do If There Is A Problem

If there are damaged or missing parts, contact your supplier and/or dealer immediately. Do not attempt to apply power to the board if there is damage to any of its components.

Table of Contents

General Description 1
Introduction 1
Feature 1
Installation 2
Switch Setting 2
Base I/O Port Address 3
Interrupt Setting 4
Jumper Setting 5
Hardware Installation 6
Programming with AX5488 Firmware Routine 7
Calling firmware routine 7
Summary of Firmware Routines 8
Firmware Routine Usage 9
(1) Function IBINIT% : Initialization 9
(2) Function IBWRT% : Write data string to device 12
(3) Function IBRD% : Read data string from device 14
(4) Function IBIFC% : Interface clear to clear GPIB bus 16
(5) Function IBEOL% : Establish an End Of Line string for I/O of a device 17
(6) Function IBDCLR% : Clear device 19
(7) Function IBLLO% : Local Lock Out to disable front panel of all devices 21
(8) Function IBLOCAL% : Make device goto local 22
(9) Function IBPPL% : Parallel Polling 24
(10) Function IBPPLC% : Parallel Polling Configure 25
(11) Function IBPPLU% : Parallel Polling Unconfigure 27
(12) Function IBREN% : Remote Enable 29
(13) Function IBCMD% : Send command code 31
(14) Function IBSPL% : Serial Polling 34
(15) Function IBSTS% : Status of GPIB bus 36
(16) Function IBTOUT% : Setup Timeout period 38
(17) Function IBTRG% : Trigger device 39
(18) Function IBRDM% : Read data string from device to Memory 41
(19) Function IBWRTM% : Write data string from Memory to device or interface 44
(20) Function IBXCHG% : Replace DOS device with GPIB device 46
(21) Function IBERR% : Monitoring error code and count of R/W string 48
Example Program 49
Programming with AX5488 Firmware Routine in C 50
Introduction 50

Link Interface Function **50**
Summary of Interface Functions **52**
Interface Functions **53**
(1) Function ibinit : Initialization 53
(2) Function ibwrt : Write data string to device 56
(3) Function ibrd : Read data string from device 58
(4) Function ibifc : Interface clear to clear GPIB bus 60
(5) Function ibeol : Establish an End Of Line string for I/O of a device 61
(6) Function ibdclr : Clear device 63
(7) Function ibllo : Local Lock out to disable front panel of all devices 65
(8) Function iblocal : Make device goto local 66
(9) Function ibppl : Parallel Polling 68
(10) Function ibpplc : Parallel Polling Configure 69
(11) Function ibpplu : Parallel Polling Unconfigure 71
(12) Function ibren : Remote Enable 73
(13) Function ibcmd : Send command code 75
(14) Function ibspl : Serial Polling 78
(15) Function ibsts : Status of GPIB bus 80
(16) Function ibtout : Setup Timeout interval 82
(17) Function ibtrg : Trigger device 83
(18) Function ibrdm : Read data string from device to Memory 85
(19) Function ibwrtm : Write data string from Memory to device or interface 87
(20) Function ibxchg : Replace DOS device with GPIB device 89
Example Program **90**

Programming with AX5488 Firmware Routine in Turbo Pascal **91**
Introduction **91**
Link Interface Function Usage **91**
Interface Functions Declaration : ift488tp.inc **92**
Summary of Interface Functions **93**
Interface Functions **94**
(1) Function ibinit : Initialization 94
(2) Function ibwrt : Write data string to device 97
(3) Function ibrd : Read data string from device 99
(4) Function ibifc : Interface clear to clear GPIB bus 101
(5) Function ibeol : Establish an End Of Line string for I/O of a device 102
(6) Function ibdclr : Clear device 104
(7) Function ibllo : Local Lock out to disable front panel of all devices 106
(8) Function iblocal : Make device goto local 107
(9) Function ibppl : Parallel Polling 109
(10) Function ibpplc : Parallel Polling Configure 110
(11) Function ibpplu : Parallel Polling Unconfigure 112

(12) Function ibren : Remote Enable 114
(13) Function ibcmd : Send command code 116
(14) Function ibspl : Serial Polling 119
(15) Function ibsts : Status of GPIB bus 121
(16) Function ibtout : Setup Timeout interval 123
(17) Function ibtrg : Trigger device 124
(18) Function ibrdm : Read data string from device to Memory 126
(19) Function ibwrtm : Write data string from Memory to device or interface 128
(20) Function ibxchg : Replace DOS device with GPIB device 130
Example Program 131
Programming with AX5488 Firmware Routine in Windows 132
Introduction 132
Link Interface Function 132
Summary of Interface Functions 136
Interface Functions 137
(1) Function ibinit : Initialization 137
(2) Function ibwrt : Write data string to device 139
(3) Function ibrd : Read data string from device 141
(4) Function ibifc : Interface clear to clear GPIB bus 143
(5) Function ibeol : Establish an End Of Line string for I/O of a device 145
(6) Function ibdclr : Clear specified device 147
(7) Function ibllo : Local Lock out to disable front panel of all devices 149
(8) Function iblocal : Make device Go To local 151
(9) Function ibppl : Parallel Polling 153
(10) Function ibpplc : Parallel Polling Configure 154
(11) Function ibpplu : Parallel Polling Unconfigure 156
(12) Function ibren : Remote Enable 158
(13) Function ibcmd : Send command code 160
(14) Function ibspl : Serial Polling 163
(15) Function ibsts : Status of GPIB bus 165
(16) Function ibtout : Setup Timeout interval 167
(17) Function ibtrg : Trigger device 169
Introduce IEEE-488 (GPIB) 171
General Introduction 171
IEEE-488 Connector Introduction 171
Interface Function 173
Handshake Timing Description 174
IEEE-488 Bus Commands and Address 175
Bus commands 175
Overview of Bus commands and address 176

Appendix A ASCII Table 178
Appendix B NEC7210 Read/Write Registers 180
Appendix C Jumpers and Switch Location 181
Appendix D Technique Reference 182
Interrupt 182
DMA 183

General Description

Introduction

The AX5488 IEEE-488 General Purpose Interface Bus (GPIB) card is designed for IBM PC/XT/AT and compatible computers slot expansion. The AX5488's hardware has a mode that is compatible with the most popular PCIIA bus structure. That allows you to run the application program written by PCIIA needs no change.

The IEEE-488 GPIB will handle up to 14 other Talker/Listener devices, the controller may be the activity PC or any other devices from the other 14 devices. The AX5488 IEEE-488 interface card is used the NECuPD7210 chip for hardware controller furnished the all IEEE-488 November 1978 standard. In addition, DMA and interrupt function is also provided.

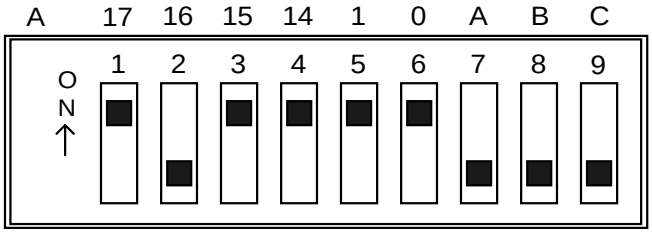
Feature

- Fully software compatible with National Instruments PCIIA.
- All GPIB ports defined according to IBM PC standard.
- Complete IEEE-488 Talker/Listener/Controller(TLC) capability using NEC7210C GPIB TLC chip.
- Powerful and simple software command set easy understanding and simple initialization.
- Data transfer rate over 300K bytes/sec.
- Half-size board fits any I/O slots or laptop PC.
- Selection of 3 DMA channels using host DMA controller.
- Choice of 6 interrupt lines.

Installation

Switch Setting

--SW1 Setting--



A,B,C : Interrupt Level

1, 0 : I/O port

17, 16, 15, 14 : Firmware address setting

Refer to next pages.

- Firmware address

  AX5488 provides IEEE-488 Interface driver Routine in the on-board EPROM. The firmware address can be selected by 4 bits. The address bits A17 to A14 map the positions 1 to 4. The address bits A19, A18 is always 1.

A	1 [17	2 16	3 15	4 14]	5	6	7	8	9	Firmware address (segment)
	ON	ON	ON	ON	X	X	X	X	X	&HC000
	ON	ON	ON	OFF	X	X	X	X	X	&HC400
	ON	ON	OFF	ON	X	X	X	X	X	&HC800
*	ON	OFF	ON	ON	X	X	X	X	X	&HD000
	:									:
	OFF	ON	ON	ON	X	X	X	X	X	&HE000
	:									:
	OFF	OFF	OFF	OFF	X	X	X	X	X	&HF000

NOTE *0 = ON, 1 = OFF, * = Factory default setting*

Base I/O Port Address

AX5488 will occupied only one I/O port space, setting correct base address by dip switch to avoid contending other device. Two bits are equipped to allow you to set the base address.

- I/O port setting

--DIP Switch Position--

	1	2	3	4	5 1	6 0	7	8	9	I/O port
*	X	X	X	X	ON	ON	X	X	X	&H02E1(GPIB0)
	X	X	X	X	ON	OFF	X	X	X	&H22E1(GPIB1)
	X	X	X	X	OFF	ON	X	X	X	&H42E1(GPIB2)
	X	X	X	X	OFF	OFF	X	X	X	&H62E1(GPIB3)

NOTE *0 = ON, 1 = OFF, * = Factory default setting*

Interrupt Setting

AX5488 provides hardware interrupt function for applications. SW1 bit 7, 8, 9 are equipped for interrupt level 2 to level 7 setting.

- Interrupt level setting

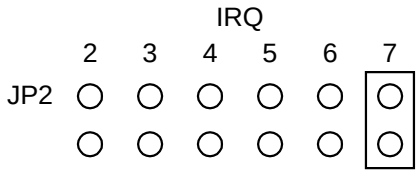
--DIP Switch Position--

	1	2	3	4	5	6	7 A	8 B	9 C	IRQ Level
	X	X	X	X	X	X	ON	ON	ON	PC Used
	X	X	X	X	X	X	ON	ON	OFF	PC Used
	X	X	X	X	X	X	ON	OFF	ON	2
	X	X	X	X	X	X	ON	OFF	OFF	3
	X	X	X	X	X	X	OFF	ON	ON	4
	X	X	X	X	X	X	OFF	ON	OFF	5
	X	X	X	X	X	X	OFF	OFF	ON	6
*	X	X	X	X	X	X	OFF	OFF	OFF	7

NOTE *0 = ON, 1 = OFF, * = Factory default setting*

Jumper Setting

- Interrupt : JP2

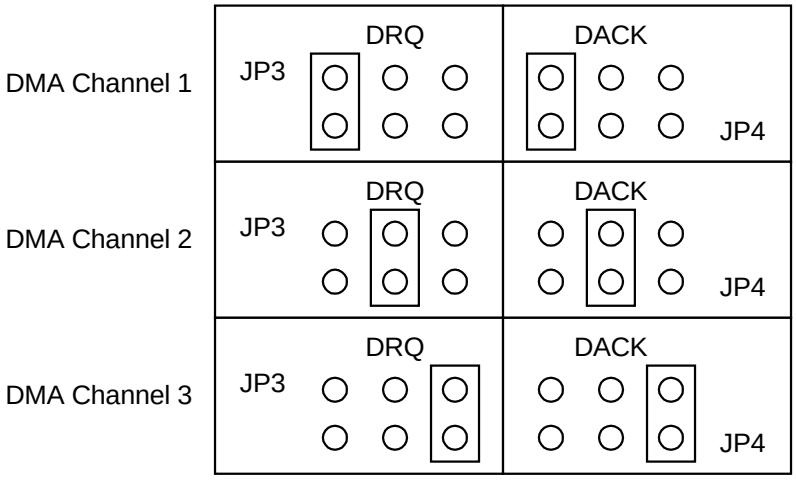


FACTORY DEFAULT SETTING AT LEVEL 7

- DMA : JP3, JP4

AX5488 also provides Direct Memory Access(DMA) capability for your application. JP3 and JP4 is established for DMA request (DRQ) and DMA acknowledge(DACK) channel setting. The acceptable DMA channel is 1, 2, or 3. Only one channel can be selected.

--Jumper Setting Diagram--



Hardware Installation

The AX5488 card is shipped with protective electrostatic cover. When unpacking, touching the board electrostatically shielded packaging with the metal frame of your computer to discharge the accumulated static electricity prior to touching the board.

Follow discussing summarizes the procedures for AX5488 installation:

WARNING	*TURN OFF the PC and all accessories connected to the PC, whenever installing or removing any peripheral board including the AX5488 series board.*

Card installation:

- Turn off the PC and all accessories power.
- Unplug all power cords and entire cables from the rear of the PC.
- Remove the PC's cover (See your PC Operation Guide if you are not skillful about it).
- Find a unused expansion slot. Remove the blank expansion slot cover and save the screw for affixing retainng bracket.
- Grab the upper edge of the AX5488 card. Align the AX5488 card's retaining bracket with the expansion slot rear panel, and straighten the board's gold finger with the expansion slot, crush the board into the slot.
- Restore the screw in the expansion slot retaining bracket.
- Replace the PC's cover and connect the cables you detached in step 2.
- Turn on the PC and other peripheral devices power.

Programming with AX5488 Firmware Routine

Calling firmware routine

The firmware routines written in assembly language is allocated in ROM with switchable segment. In Basic and Basic compiler language, you can call desired routine according to the associated offset, named as mnemonic words IBINIT%, IBWRT% and etc. that can be defined as you wish. Before calling routine, the ROM segment and routine offset from this segment must be defined previously, then associated parameter is needed to make the routine work properly.

An example is given as follow.

```
*** BASICA ***
DEF SEG = &HD000              ' ROM SEGMENT = &HD000
IBINIT% = 0                   ' OFFSET = 0
IOPORT% = &H2E1               ' PARAMETER
MYADDR% = 21                  ' PARAMETER
CONFIG% = &H008F              ' PARAMETER
CALL IBINIT% (IOPORT%, MYADDR%, CONFIG%)

*** BASICA COMPILER ***
DEF SEG = &HD000              ' ROM SEGMENT = &HD000
IBINIT% = 0                   ' OFFSET = 0
IOPORT% = &H2E1               ' PARAMETER
MYADDR% = 21                  ' PARAMETER
CONFIG% = &H108F              ' PARAMETER
CALL ABSOLUTE (IOPORT%, MYADDR%, CONFIG%, IBINIT%)
```

> REMARK *All of offset, parameters must be variable instead of constant, because Basic pass their address in the sequence written to Basic's stack, on executing the CALL.*

Summary of Firmware Routines

Mnemonic	Offset	Description
IBINIT%	0	Initialize the AX5488 driver function routine
IBWRT%	3	Write data string to device
IBRD%	6	Read data string from device
IBIFC%	9	Interface clear to reset GPIB bus
IBEOL%	12	Establish an End Of Line string for I/O of a device
IBDCLR%	15	Clear specified Device
IBLLO%	18	Local lockout to disable device front panel operation
IBLOCAL%	21	Make device go to local
IBPPL%	24	Parallel polling
IBPPLC%	27	Parallel polling configure
IBPPLU%	30	Parallel polling unconfigure
IBREN%	33	Set device to remote mode by setting Remote Enable line
IBCMD%	36	Send command string to device
IBSPL%	39	Serial polling and return device's status
IBSTS%	42	Status of the GPIB bus
IBTOUT%	45	Setup time out interval
IBTRG%	48	Trigger specified device
IBRDM%	51	Read data into memory with specified segment
IBWRTM%	54	Write data from memory to device
IBXCHG%	57	Replace DOS device with GPIB device
IBERR%	60	Return error code

Firmware Routine Usage

(1) Function IBINIT% : Initialization

- Introduction

  1. Setup base address.
  2. Setup parameter to make GPIB controller work properly.
  3. Setup interrupt level and DMA channel.

  This function needed to be executed only once, must be performed before any other functions.

- Format

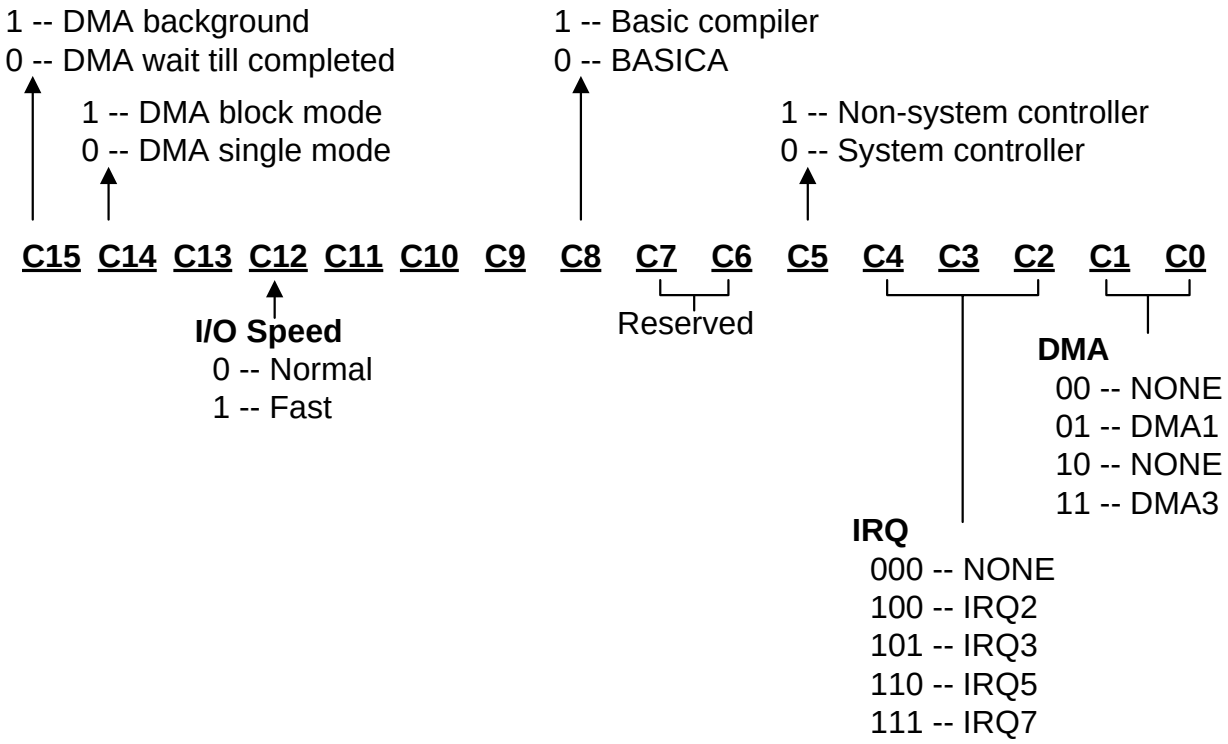
```
*** BASICA ***
IBINIT% = 0       ‘ OFFSET
CALL    IBINIT% (IOPORT%, GPIBADDR%, CONFIG%)

*** BASICA COMPILER***
IBINIT% = 0       ‘ OFFSET
CALL    ABSOLUTE (IOPORT%, GPIBADDR%, CONFIG%, IBINIT%)
```

- Parameter

  IOPORT% – I/O port of GPIB controller NEC7210, default 2E1 hex..

  GPIBADDR% – GPIB address of device, 0 through 30, default 21.

  CONFIG% – Configure of the firmware routine, depending on setting bit of 16, C15…C0.



The following setting are for Basic language only.

C9 : 0 – None
1 – When SRQ happen, ON KEY 20 active

C10 : 0 – None
1 – When Timeout happen, ON KEY 19 active

C11 : 0 – None
1 – When Error happen, ON ERROR active

C13 : Determine whether it is in Basic interpreter or compiler for setting of C9 through C11.
0 – for Basic interpreter
1 – for Basic compiler

- GPIB bus operation

None.

- Example

  :

  Set AX5488 operation mode as follows:

  Basic interpreter, System controller, DMA channel 3 & single transfer mode, GPIB address 21, I/O port hex. 2E1.

```
10  DEF SEG = &HD000          ' Firmware segment
20  IBINIT% = 0               ' offset = 0
30  GPIBADDR% = 21            ' GPIB address = 21
40  IOPORT% = &H2E1           ' I/O port = 2E1 hex.
50  CONFIG% = &H0003          ' bit 0, 1 are all 1.
60  CALL IBINIT% (IOPORT%, GPIBADDR%, CONFIG%)
```

(2) Function IBWRT% : Write data string to device

- Introduction

  The routine write a string followed a terminator to specified device.

- Format

```
*** BASICA ***
IBWRT% = 3              ' OFFSET
CALL    IBWRT% (GPIBADDR%, WRTSTR$)

*** BASIC COMPILER ***
IBWRT% = 3              ' OFFSET
CALL    ABSOLUTE(GPIBADDR%, WRTSTR$, IBWRT%)
```

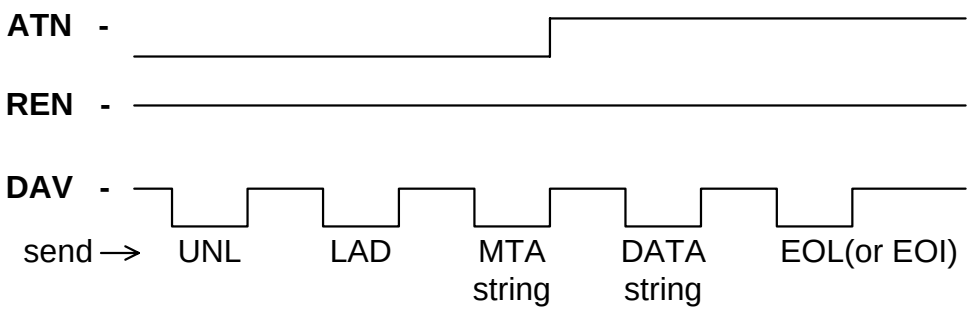
- Parameter

  GPIBADDR% – GPIB address of device, 0 through 30.

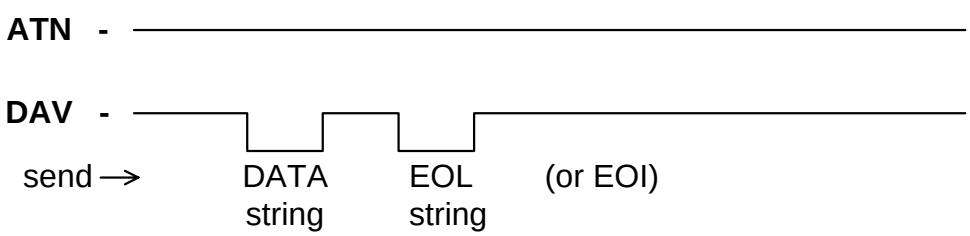
  WRTSTR$ – Output data string.

- GPIB bus operation : Low -- true, High -- false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

  :

  Operation :

  Write a string to Hp3478, address 23.

```
10  DEF SEG = &HD000          ‘ Firmware segment
20  IBWRT% = 3                ‘ Offset = 3
30  GPIBADDR% = 23            ‘ GPIB address = 23
40  WRTSTR$ = “F1T3RAN3”      ‘ Set up the HP3478
50  CALL IBWRT% (GPIBADDR%, WRTSTR$)
```

(3) Function IBRD% : Read data string from device

- Introduction

  The routine read a string followed a terminator to specified device.

- Format

```
*** BASICA ***
IBRD% = 6                ' OFFSET
CALL    IBRD%(GPIBADDR%, RDSTR$)

*** BASIC COMPILER ***
IBRD% = 6                ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, RDSTR$, IBRD%)
```

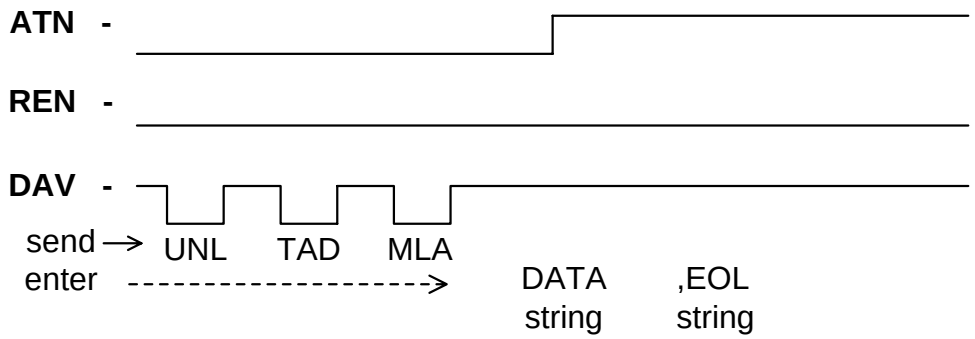
- Parameter

  GPIBADDR% – GPIB address of device, 0 through 30.

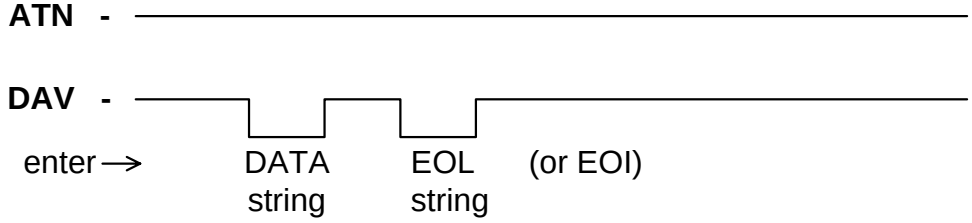
  RDSTR$ – Input data string.

- GPIB bus operation : Low -- true, High -- false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

:

Operation :

Read a string from Hp3478, address 23.

```
10   DEF SEG = &HD000          ' Firmware segment
20   IBRD% = 6                 ' offset = 6
30   GPIBADDR% = 23            ' GPIB address = 23
40   RDSTR$ = "          "     ' Set up the HP3478
50   CALL IBRD% (GPIBADDR%, RDSTR$)
60   PRINT "READ DATA = ", VAL(RDSTR$)
```

(4) Function IBIFC% : Interface clear to clear GPIB bus

- Introduction

  This routine, sending a IFC signal to clear GPIB interface bus, can be called by system controller only.

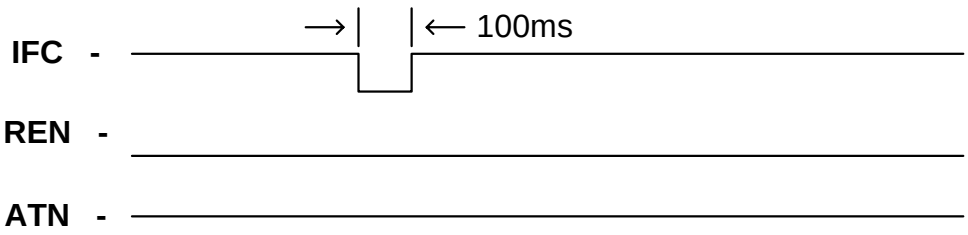
- Format

```
*** BASICA ***
IBIFC% = 9              ' OFFSET
CALL    IBIFC%

*** BASIC COMPLIER ***
IBIFC% = 9              ' OFFSET
CALL    ABSOLUTE(IBIFC%)
```

- Parameter

  NONE

- GPIB bus operation : Low -- true, High -- false



- Example

  :

  Operation :

  Clear all device

```
10    DEF SEG = &HD000        ' Firmware segment
20    IBIFC% = 9              ' offset = 9
30    CALL IBIFC%
```

(5) Function IBEOL% : Establish an End Of Line string for I/O of a device

- Introduction

  Use this routine to establish an end of line terminator of output and input strings for specified device, otherwise the default terminators are set.
  Output terminator is default CR (13), LF (10).
  Input terminator is default LF (10).

- Format

  *** BASICA ***

  IBEOL% = 12 ' OFFSET

  CALL IBEOL%(GPIBADDR%, OMODE%, OEOL$, IMODE%, IEOL%)

  *** BASIC COMPILER ***

  IBEOL% = 12 ' OFFSET

  CALL ABSOLUTE (GPIBADDR%, OMODE%, OEOL$, IMODE%, IEOL%, IBEOL%)

- Parameter

  GPIBADDR% – GPIB address of device, 0 through 30.

  OMODE$ – Output terminator mode (0-2), default 0.
  0 : Terminated when encounter OEOL$ or EOI is active.
  1 : Terminated when EOI is active.
  2 : Terminated when encounter OEOL$.

  OEOL$ – Output End Of Line string following output string can be set up to 8 characters, default CR (13) and LF (10).

  IMODE$ – Input terminator mode (0-1), default 0.
  0 : Terminated when encounter IEOL% or EOI is active or input string full.
  1 : Terminated when EOI is true or input string full.

  IEOL% – Input End Of Line character, default LF(10).

- GPIB bus operation

  NONE

- Example

:

Operation :

Set hp3478 terminator, addressed 23.

```
10  DEF SEG = &HD000                          ' Firmware segment
20  IBEOL% = 12                               ' offset = 12
30  GPIBADDR% = 23                            ' GPIB address = 23
40  OMODE% = 0
50  OEOL$ = CHAR$(13) + CHAR$(10)             ' Output terminator CR, LF
60  IMODE% = 0
70  IEOL% = 10                                ' Input terminator LF
80  CALL IBEOL% (GPIBADDR%, OMODE%, OEOL$, IMODE%,
    IEOL%)
```

(6) Function IBDCLR% : Clear device

- Introduction

  Send SDC command to clear specified device or send DCL command to clear all devices.

- Format

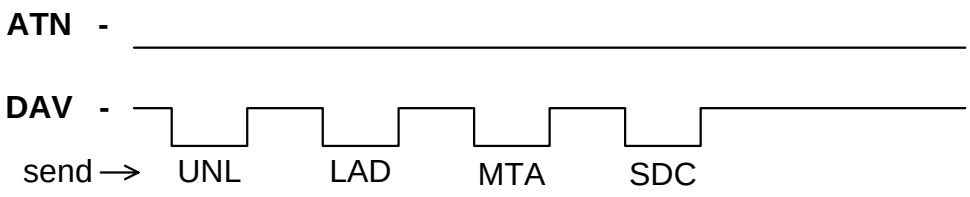
```
*** BASICA ***
IBDCLR% = 15                ' OFFSET
CALL    IBDCLR%(GPIBADDR %)

*** BASIC COMPILER ***
IBDCLR% = 15                ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, IBDCLR%)
```

- Parameter

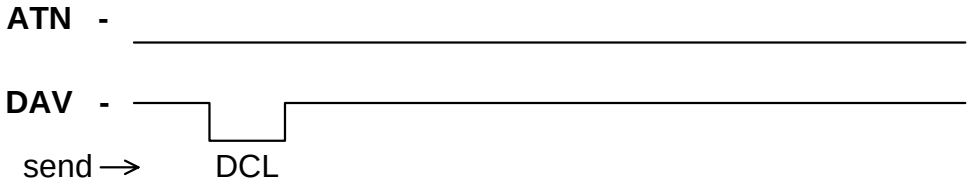
  GPIBADDR% – GPIB address of device, 0 through 30.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

:

Operation :

Clear hp3478, addressed 23.

```
10    DEF SEG = &HD000              ' Firmware segment
20    IBDCLR% = 15                  ' offset = 15
30    GPIBADDR% = 23                ' GPIB address = 23
40    CALL IBDCLR% (GPIBADDR%)
```

(7) Function IBLLO% : Local Lock Out to disable front panel of all devices

- Introduction

  Send LLO command to disable front panel of all devices.

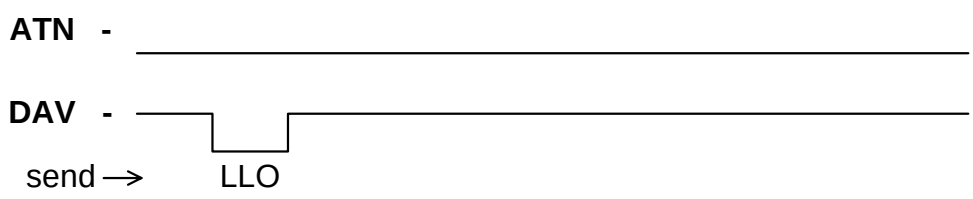
- Format

```
*** BASICA ***
IBLLO% = 18              ' OFFSET
CALL    IBLLO%

*** BASIC COMPILER ***
IBLLO% = 18              ' OFFSET
CALL    ABSOLUTE (IBLLO%)
```

- Parameter

  NONE

- GPIB bus operation : Low – true, High – false



- Example

```
  :
Operation :
Disable front panel of all devices
  10    DEF SEG = &HD000          ' Firmware segment
  20    IBLLO% = 18               ' offset = 18
  30    CALL IBLLO%
```

(8) Function IBLOCAL% : Make device goto local

- Introduction

  Sent Go To Local (GTL) command or set REN false to enable front panel of specified device.

- Format

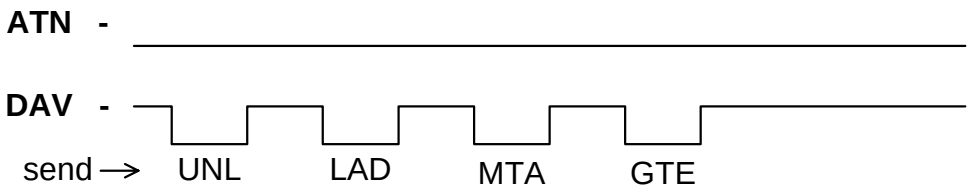
```
*** BASICA ***
IBLOCAL% = 21              ' OFFSET
CALL    IBLOCAL%(GPIBADDR%)

*** BASIC COMPILER ***
IBLOCAL% = 21              ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, IBLOCAL%)
```

- Parameter

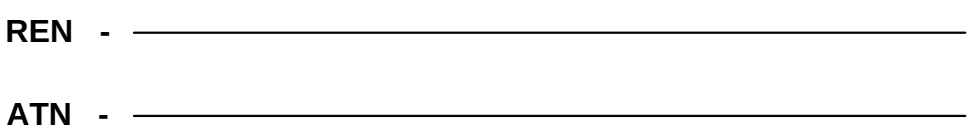
  GPIBADDR% – GPIB address of device, 0 through 30.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

:

Operation :

Enable hp3478's front panel.

```
10    DEF SEG = &HD000          ' Firmware segment
20    IBLOCAL% = 21             ' offset = 21
30    GPIBADDR% = 23            ' GPIB address = 23
40    CALL IBLOCAL% (GPIBADDR%)
```

(9) Function IBPPL% : Parallel Polling

- Introduction

  The Parallel polling routine allows system controller to check up to 8 devices simultaneously with SRQCHK%. If any device requests service the associated bit of DIO bus is setup. After calling this routine, the DIO bus data is then transferred to SRQCHK%.

  Refer to next routine IBPPLC% for DIO bit definition.

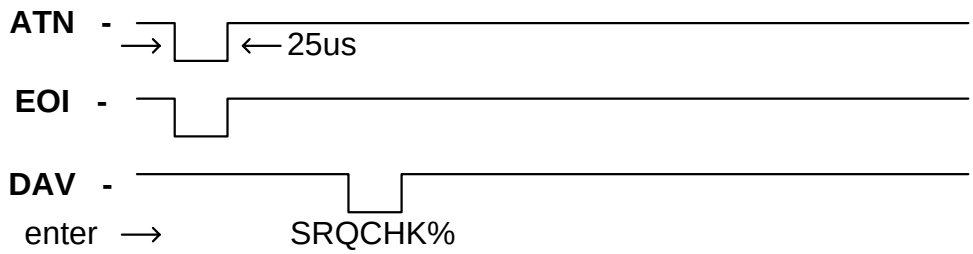
- Format

```
*** BASICA ***
IBPPL% = 24            ' OFFSET
CALL   IBPPL%(SRQCHK%)

*** BASIC COMPILER ***
IBPPL% = 24            ' OFFSET
CALL   ABSOLUTE (SRQCHK%, IBPPL%)
```

- Parameter

  SRQCHK% – Return DIO bus data to SRQCHK% integer after parallel polling. Only low byte makes sense.

- GPIB bus operation : Low – true, High – false



- Example

```
:
10   DEF SEG = &HD000          ' Firmware segment
20   IBPPL% = 24               ' offset = 24
30   CALL IBPPL% (SRQCHK%)
40   PRINT USING "SRQCHK=\ \"; HEX$(SRQCHK%)
```

(10) Function IBPPLC% : Parallel Polling Configure

- Introduction

  The Parallel polling Configure routine allow device to specify a DIO bus line to respond to parallel poll. It uses integer CONFIG% bit 0 through 2 to select DIO bus line, bit 3 to define the response state of that line.

- Format

```
*** BASICA ***
IBPPLC% = 27                ' OFFSET
CALL    IBPPLC%(GPIBADDR%, CONFIG%)

*** BASIC COMPILER ***
IBPPLC% = 27                ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, CONFIG%, IBPPLC%)
```

- Parameter

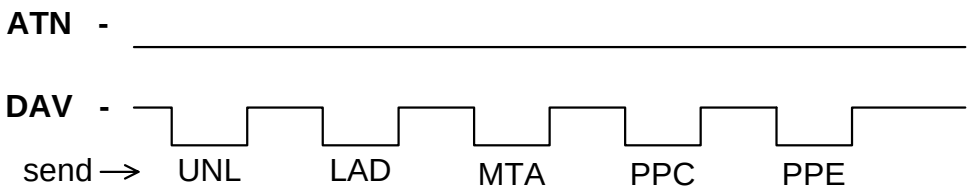
  GPIBADDR% – GPIB address of device, 0 through 30.

  CONFIG% – An integer to configure a device to respond to parallel polling. Only the low nibble make sense.

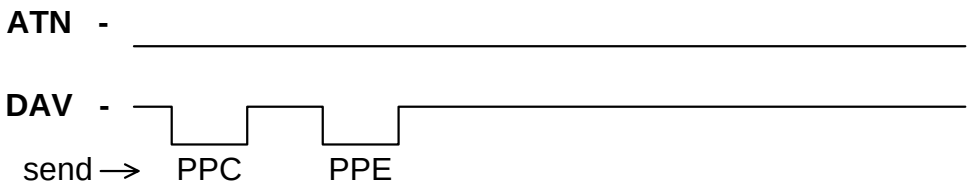
```
BIT 3      2    1    0
           |
000 – DIO1
001 – DIO2
    :
007 – DIO8
```

  BIT 3 – Determine the state of data line(DIO#).
  0 : low state
  1 : high state

- GPIB bus operation : Low – true, High – false
  - ✧ Legal GPIB address



  - ✧ Illegal GPIB address



- Example

:

Operation :

Configure hp3478, addressed 23.

```
10    DEF SEG = &HD000   ‘ Firmware segment
20    IBPPLC% = 27       ‘ offset = 27
30    CONFIG% = &H00     ‘ select DIO0, reflect with low state
40    GPIBADDR% = 23     ‘ gpib address = 23
50    CALL IBPPLC% (GPIBADDR%, CONFIG%)
```

(11) Function IBPPLU% : Parallel Polling Unconfigure

- Introduction

  The Parallel polling Unconfigure routine is used to tell specified device or all device on the interface bus not to respond to the parallel polling. If GPIBADDR% is legal, the addressed device will be unconfigured, otherwise all devices will be unconfigured.

- Format

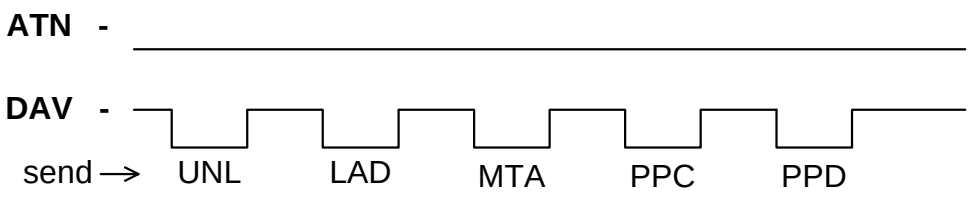
```
*** BASICA ***
IBPPLU% = 30                ' OFFSET
CALL    IBPPLU%(GPIBADDR%)

*** BASIC COMPILER ***
IBPPLU% = 30                ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, IBPPLU%)
```

- Parameter

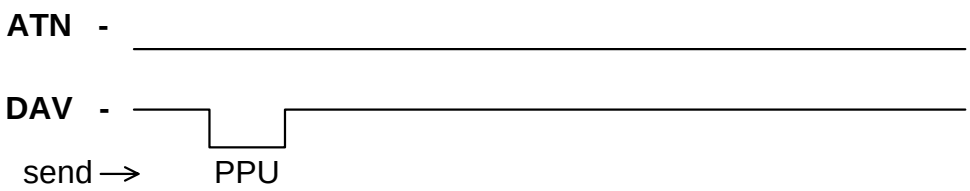
  GPIBADDR% – GPIB address of device, 0 through 30.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

:

Operation :

Unconfigure hp3478, addressed 23.

```
10    DEF SEG = &HD000              ‘ Firmware segment
20    IBPPLU% = 30                  ‘ offset = 30
30    GPIBADDR% = 23                ‘ gpib address = 23
40    CALL IBPPLU% (GPIBADDR%)
```

(12) Function IBREN% : Remote Enable

- Introduction

  This routine is used set specified device or all devices on the interface bus to remote mode. If GPIBADDR% is legal, the addressed device will be in remote mode, otherwise all devices will be in remote mode.

- Format

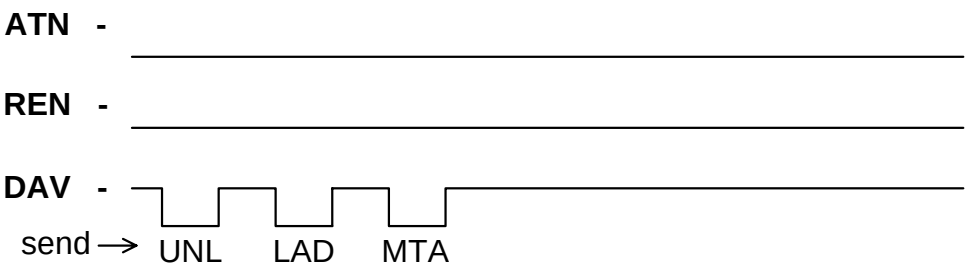
```
*** BASICA ***
IBREN% = 33                ' OFFSET
CALL    IBREN%(GPIBADDR%)

*** BASIC COMPILER ***
IBREN% = 33                ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, IBREN%)
```

- Parameter

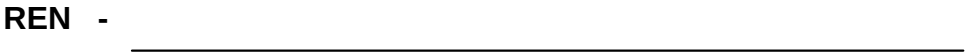
  GPIBADDR% – GPIB address of device, 0 through 30.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address

  ✧ Illegal GPIB address

- Example

:

Operation :

Set hp3478's addressed 23 to remote mode.

```
10  DEF SEG = &HD000            ' Firmware segment
20  IBREN% = 33                 ' offset = 33
30  GPIBADDR% = 23              ' gpib address = 23
40  CALL IBREN% (GPIBADDR%)
```

(13) Function IBCMD% : Send command code

- Introduction

  To send command direct to gpib interface bus, or to control some devices simultaneously, use this routine.

- Format

```
*** BASICA ***
IBCMD% = 36                ' OFFSET
CALL    IBCMD%(CMD$)

*** BASIC COMPILER ***
IBCMD% = 36                ' OFFSET
CALL    ABSOLUTE (CMD$, IBCMD%)
```

- Parameter

  GCMD% – GPIB bus command string sent to GPIB bus. There are two group commands, one is bus command sending character to the bus, the other is action command which require some actions from GPIB controller.

✧ Bus commands are listed as follows:

Mnemonic	Hex.	Mode	Description
GTL	01	Address	Goto Local. Set addressed device to local mode.
SDC	04	Address	Selected Device Clear. Reset addressed device.
PPC	05	Address	Parallel Poll Configure.
GET	08	Address	Group Execute Trigger. Trigger all listeners on the bus.
TCT	09	Global	Take Control. Transfer control to another device.
LLO	11	Global	Local Lockout to disable front panel of all devices.
DCL	14	Global	Device Clear
PPU	15	Global	Parallel Poll Unconfigure.

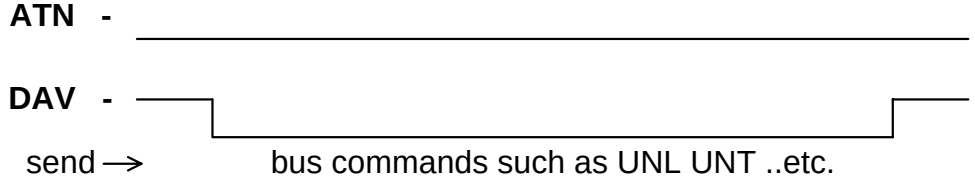
Mnemonic	Hex.	Mode	Description
SPE	18	Global	Serial Poll Enable.
SPD	19	Global	Serial Poll Disable.
UNL	3F	Address	Unlisted. Unaddress the listener.
UNT	5F	Address	Untalk. Unaddress the talker.

REMARK *Global means that the commands are sent to all devices on the bus. Address means that the commands are sent to address device only.*

✧ Action commands are listed as follows:

Mnemonic	Description
LISTEN	Values following this word are listener addresses.
TALK	Value following this word is talker address.
SEC	Values following this word are secondary commands
PPE	Values following this word are parallel poll configure.
DATA	Set ATN false for data mode.
CMD	Set ATN true for command mode.
IFC	Send IFC pulse for 100ms.
EOI	Set EOI true after the last byte of output string is sent.

- GPIB bus operation : Low – true, High – false



- Example

:

Operation :

Send bus commands to trigger hp3478, addressed 23.

```
10    DEF SEG = &HD000              ‘ Firmware segment
20    IBCMD% = 36                   ‘ offset = 36
30    CMD$ = “UNL UNT MTA LISTEN 23 GET”;
40    CALL IBCMD% (CMD$)
```

(14) Function IBSPL% : Serial Polling

- Introduction

  The Serial polling routine allows system controller to know whether the specified device request service by checking the returned variable SRQCHK% low byte.

- Format

```
*** BASICA ***
IBSPL% = 39            ' OFFSET
CALL    IBSPL%(GPIBADDR%, SRQCHK%)

*** BASIC COMPILER ***
IBSPL% = 39            ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, SRQCHK%, IBSPL%)
```

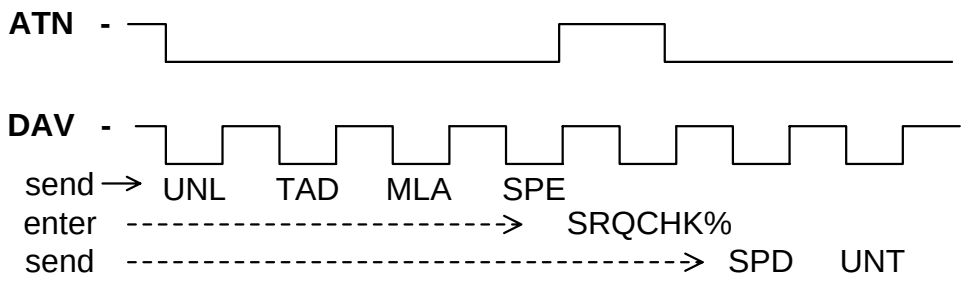
- Parameter

  GPIBADDR% – GPIB address of device, 0 through 30.

  SRQCHK% – Return DIO bus data to SRQCHK% integer after serial polling. Only low byte makes sense.

```
BIT   7   6   5   4   3   2   1   0
          |
        SRQ
      0 : Active
      1 : None
Bit 7 : reserved
Bit 1 through 5 are defined depending on device.
```

- GPIB bus operation : Low – true, High – false



- Example

:

Operation :

Serial poll hp3478, addressed 23 after triggered to determine whether the data is ready to be read by checking return variable SRQCHK% bit 0.

```
10   DEF SEG = &HD000              ‘ Firmware segment
20   IBSPL% = 39                   ‘ offset = 39
30   GPIBADDR% = 23                ‘ HP3478 addressed 23
40   CALL IBSPL% (GPIBADDR%, SRQCHK%)
50   IF (SRQCHK% AND &H01) = &H01 THEN GOSUB
     READ3478DATA
```

(15) Function IBSTS% : Status of GPIB bus

- Introduction

  This routine allows you to monitor the status of GPIB bus, NEC7210 controller and etc. by checking the returned variable STATUS%.

- Format

```
*** BASICA ***
IBSTS% = 42              ' OFFSET
CALL    IBSTS%(CFG%, STATUS%)

*** BASIC COMPILER ***
IBSTS% = 42              ' OFFSET
CALL    ABSOLUTE (CFG%, STATUS%, IBSTS%)
```

- Parameter

  CFG% – A 16 bit configure variable to determine what status to be returned.

0 – 7	:	Return NEC7210 read register 0 – 7.
8	:	Return error number of last call routine.
9	:	Return length of output or input string.
10	:	Return Timeout period.
11	:	Return I/O port of NEC7210.
12	:	Return DMA and IRQ setting.
13 – 15	:	None

REMARK *If bit 8 is set, then error NO. is returned as follows.*

Error No.	Description
0	No error
1	Handshake timeout
2	Interface error
3	Non-system controller call IBIFC%
4	Parameter setting error

STATUS% – Returned status variable depending on CFG% variable setting.

- GPIB bus operation :

  None.

- Example

  :

  Operation :

  Read error number of last call routine.

```
10  DEF SEG = &HD000        ' Firmware segment
20  IBSTS% = 42             ' offset = 42
30  CFG% = &H10             ' return error NO. last routine
40  CALL IBSTS% (CFG%, STATUS%)
50  PRINT USING "ERROR NO. = ##"; STATUS%
```

(16) Function IBTOUT% : Setup Timeout period

- Introduction

  This routine allows you to set timeout period. If there is any malfunction in handshake, then the interface line is stuck until timeout period is up.

- Format

  *** BASICA ***

  IBTOUT% = 45 ‘ OFFSET

  CALL IBTOUT%(INTVL%)

  *** BASIC COMPILER ***

  IBTOUT% = 45 ‘ OFFSET

  CALL ABSOLUTE (INTVL%, IBTOUT%)

- Parameter

  INTVL% – Timeout period (1 – 65535).
  0 : Disable timeout routine.
  N : N units N = 1 to 32767
  N : N + 65535 units N = -1 to -32767

- GPIB bus operation

  None

- Example

  :

  Operation :

  Set timeout period for gpib interface bus.

```
10  DEF SEG = &HD000        ‘ Firmware segment
20  IBTOUT% = 45            ‘ offset = 45
30  INTVL% = 100            ‘ timeout period 100 units
40  CALL IBTOUT% (INTVL%)
```

(17) Function IBTRG% : Trigger device

- Introduction

  This routine is used to trigger specified device or all devices on the interface bus by sending Group Execute Trigger (GTE) to the interface bus. If GPIBADDR% is legal, the addressed device will be triggered, otherwise all devices will be triggered.

- Format

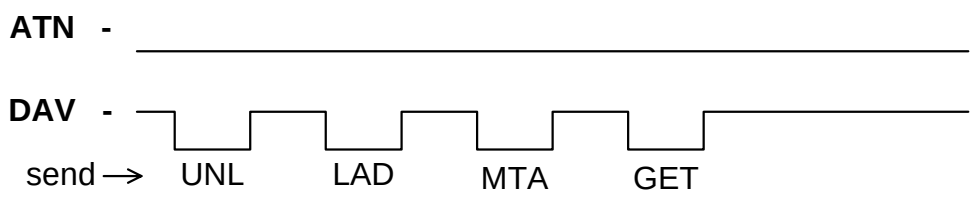
```
*** BASICA ***
IBTRG% = 48                ' OFFSET
CALL    IBTRG%(GPIBADDR%)

*** BASIC COMPILER ***
IBTRG% = 48                ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, IBTRG%)
```

- Parameter

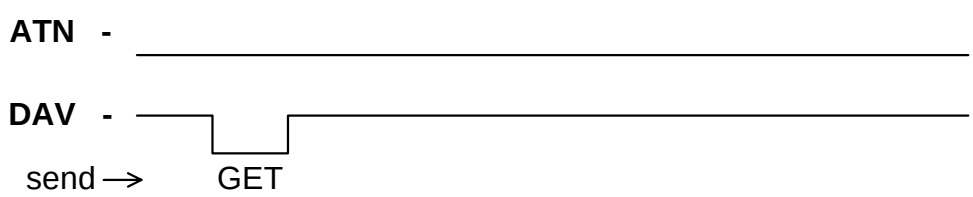
  GPIBADDR% – GPIB address of device, 0 through 30.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

:

Operation :

Trigger hp3478's addressed 23.

```
10    DEF SEG = &HD000              ' Firmware segment
20    IBTRG% = 48                   ' offset = 48
30    GPIBADDR% = 23                ' gpib address = 23
40    CALL IBTRG% (GPIBADDR%)
```

(18) Function IBRDM% : Read data string from device to Memory

- Introduction

  This routine enables you to read a long data string up to 64K bytes from a device or interface bus into memory buffer with specified segment. Data string will be always placed from offset 0.

  The following condition will terminate the reading :

  - Reading number is reached.
  - Terminator or EOI is encountered.
  - Timeout.

- Format

  *** BASICA ***

  IBRDM% = 51 ‘ OFFSET

  CALL IBRDM%(GPIBADDR%, MSEG%, LEN%)

  *** BASIC COMPILER ***

  IBRDM% = 51 ‘ OFFSET

  CALL ABSOLUTE (GPIBADDR%, MSEG%, LEN%, IBRDM%)

- Parameter

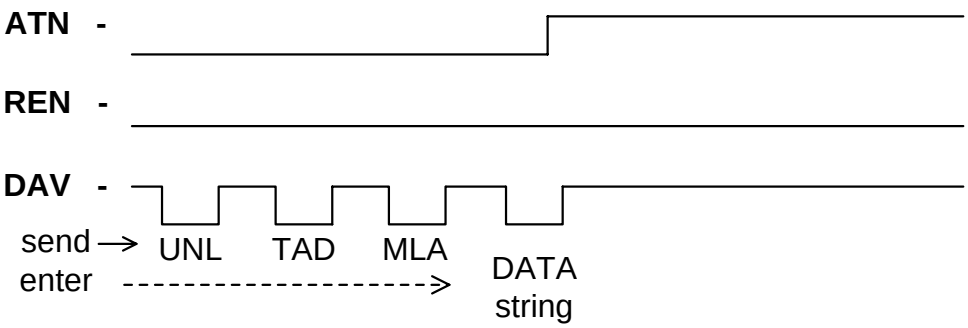
  GPIBADDR% – GPIB address of device, 0 through 30.

  MSEG% – Segment of specified memory buffer. Be sure that the segment is not in use by your program.

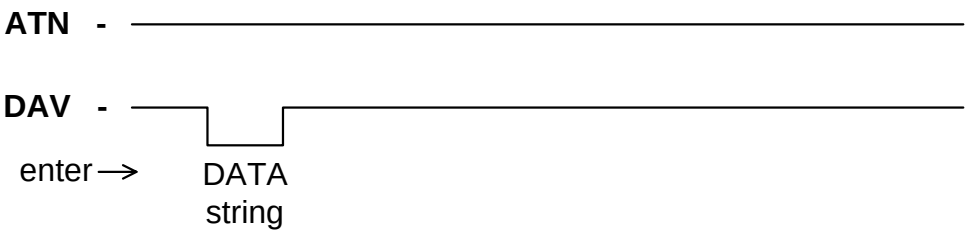
  The offset starts from 0 to 65535.

  LEN% – Length of string 0 through 65535.

- GPIB bus operation : Low – true, High – false
  - ✧ Legal GPIB address



  - ✧ Illegal GPIB address



- Example

:

Operation :

Read a 1024 bytes of string from Hp3478, addressed 23.

```
10   DEF SEG = &HD000          ‘ Firmware segment = &HD000
20   IBRDM% = 51               ‘ offset = 51
30   IBSTATUS% = 42            ‘ offset = 42
40   GPIBADDR% = 23            ‘ GPIB address = 23
50   MSEG% = &H6000            ‘ Memory segment = &h6000
60   LEN% = 1024               ‘ Collect 1024 bytes data
70   CALL IBRDM%               ‘ Collect data now
     (GPIBADDR%, MSEG%,
     LEN%)
80   CFG% = 9                  ‘ Set configure = 9 for reading
                                 number
90   RDNO% = 0                 ‘ Reset reading number
100  CALL IBSTATUS%(CFG%,      ‘ Read reading number now
     RDNO%)
110  DEF SEG = &H6000          ‘ Define memory segment for
                                 getting data
120  IF RDNO% >=0 THEN RDNUMBER = RDNO% ELSE
     RDNUMBER = 65535! + RDNO%
130  FOR I = 0 TO RDNUMBER   - 1
140       PRINT SHR$(PEEK(I));
150  NEXT I
160  DEF SEG = &HD000          ‘ Re-define firmware segment
                                 for another ‘ calling routine
```

(19) Function IBWRTM% : Write data string from Memory to device or interface

- Introduction

  This routine enables you to write a long data string up to 64K bytes from memory buffer with specified segment to a device or interface bus. The starting offset of the data memory buffer is 0.

  The following condition will terminate the writing:

  - Writing number is reached.
  - Timeout.

- Format

```
*** BASICA ***
IBWRTM% = 54              ' OFFSET
CALL    IBWRTM%(GPIBADDR%, MSEG%, LEN%)

*** BASIC COMPILER ***
IBWRTM% = 54              ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, MSEG%, LEN%, IBWRTM%)
```

- Parameter

  GPIBADDR% – GPIB address of device, 0 through 30.

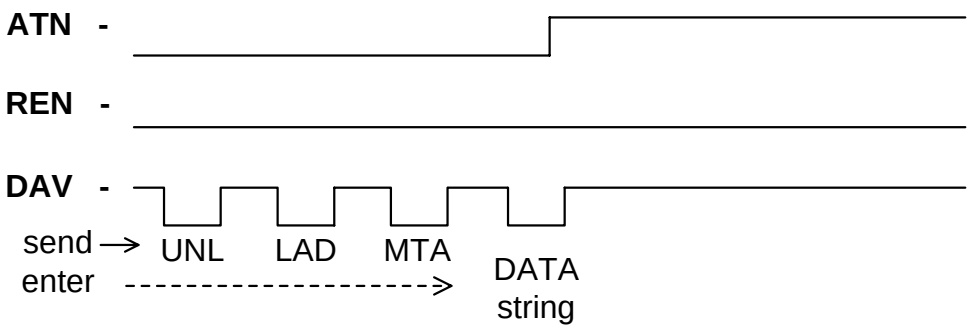
  MSEG% – Segment of specified memory buffer. Be sure that the segment is not in use by your program.

  The offset starts from 0 to 65535.

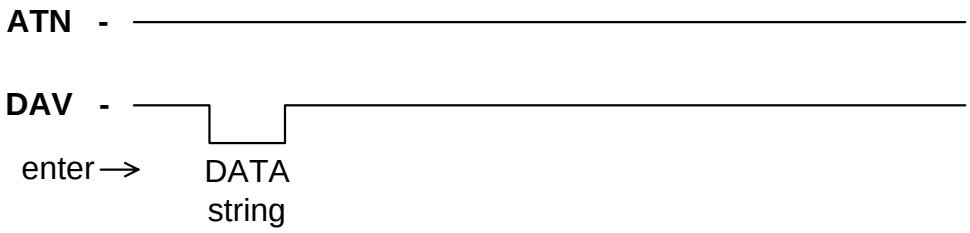
  LEN% – Length of string 0 through 65535.

- GPIB bus operation : Low – true, High – false

  ✧ **Legal GPIB address**



  ✧ **Illegal GPIB address**



- Example

  :

  Operation :

  Write a 1024 bytes of string allocated in memory with segment &h6000 to Hp3478, addressed 23.

```
10  DEF SEG = &HD000          ' Firmware segment = &HD000
20  IBWRTM% = 54              ' offset = 54
30  GPIBADDR% = 23            ' GPIB address = 23
40  MSEG% = &H6000            ' Memory segment = &h 6000
50  LEN% = 1024               ' Write 1024 bytes data
60  CALL IBWRTM%              ' Write data now
    (GPIBADDR%, MSEG%,
    LEN%)
```

(20) Function IBXCHG% : Replace DOS device with GPIB device

- Introduction

  This routine enables you to replace DOS device with GPIB device. Then, you can access the GPIB device using DOS commands. The DOS devices are LPT1:, LPT2:, LPT3:, COM1: and COM2:.

- Format

  *** BASICA ***

  IBXCHG% = 57 ‘ OFFSET

  CALL IBXCHG%(GPIBADDR%, UNIT%)

  *** BASIC COMPILER ***

  IBXCHG% = 57 ‘ OFFSET

  CALL ABSOLUTE (GPIBADDR%, UNIT%, IBWRTM%)

- Parameter

  GPIBADDR% – GPIB address of device, 0 through 30.

  UNIT% – Define which device is to be replaced with GPIB device.
  1 – LPT1:
  2 – LPT2:
  3 – LPT3:
  4 – COM1:
  5 – COM2:

- GPIB bus operation

  None

- Example

:

Operation :

Install GPIB printer, addressed 8 in place of DOS LPT1:

```
10  DEF SEG = &HD000          ‘ Firmware segment = &hc000
20  IBXCHG% = 57              ‘ offset = 57
30  GPIBADDR% = 8             ‘ GPIB address = 8
40  UNIT% = 1                 ‘ LPT1:
50  CALL IBXCHG%              ‘ Change now
    (GPIBADDR%, UNIT%)
60  OPEN “LPT1:’ FOR OUTPUT AS #1
70  PRINT #1, “A”             ‘ output character A to GPIB printer
80  CLOSE #1
```

(21) Function IBERR% : Monitoring error code and count of R/W string

- Introduction

  This routine enables you to assign two variables to monitor the operation of interface bus. It must be executed before any other routines except IBINIT% routine. The two variables are ERRNO% and RWCOUNT%.

Error No.	Description
0	No error
1	Handshake timeout
2	Interface error
3	Non-system controller call IBIFC%
4	Parameter setting error

- Format

```
*** BASICA ***
IBERR% = 60                ' OFFSET
CALL    IBERR%(GPIBADDR%, UNIT%)

*** BASIC COMPILER ***
IBERR% = 60                ' OFFSET
CALL    ABSOLUTE (GPIBADDR%, UNIT%, IBERR%)
```

- Parameter

  ERRNO% – Error number of GPIB interface bus.

Error No.	Description
0	No error
1	Handshake timeout
2	Interface error
3	Non-system controller call IBIFC%
4	Parameter setting error

  RWCOUNT% – Return the current count of read or write string.

- GPIB bus operation

  None

- Example

:

Operation :

Setup variables ERRNO% & RWCOUNT% to monitor interface status.

```
10   DEF SEG = &HD000          ' Firmware segment = &hc000
20   IBWRT% = 3                ' offset = 3
30   IBRD% = 6                 ' offset = 6
40   IBERR% = 60               ' offset = 60
50   GPIBADDR% = 23            ' GPIB address = 23
60   CALL IBERR%(ERRNO%, RWCOUNT%)
70   WRTSTR$ = "F1T3RAN3"      ' Set up the HP3478
80   CALL IBWRT%(GPIBADDR%, WRTSTR$)
90   GOSUB 150
100  RDSTR$ = SPACE$(80)
110  CALL IBRD%(GPIBADDR%, RDSTR$)
120  GOSUB 150
130  PRINT RDSTR$
140  END
150  PRINT USING "RWCOUNT = ##### BYTES";RWCOUNT%
160  IF ERRNO% = 0 THEN PRINT "NO ERROR"
170  IF ERRNO% = 1 THEN PRINT "HANDSHAKE TIMEOUT"
180  IF ERRNO% = 2 THEN PRINT "INTERFACE ERROR"
190  RETURN
```

Example Program

- **DMM.BAS :** *Voltage Measurement with DMM (HP3478)*
- **SENDCMD.BAS :** *Voltage Measurement with DMM (HP3478) by sending commend string to interface bus.*
- **AD500.BAS :** ***R****esistance Measurement with DMM (HP3478) and AD500 programmable measurement unit.*

Programming with AX5488 Firmware Routine in C

Introduction

In Basic language, it can be called directly by define segment and offset, but for other language, it cannot work by calling this way. So, we develop interface function for you to call the firmware routine. These interface functions support the following languages:

- Microsoft C
- Borland Turbo C

Link Interface Function

This driver having to be linked while compiling user's programs, is just prepared for Microsoft C and Turbo C both. The following examples show you how to link it in small model.

- Microsoft C

  a. cl /c /AS dmm.c → produce dmm.obj

  b. link → produce dmm.exe

  [.obj] : dmm.obj

  [.exe] : dmm.exe

  [.map] :

  [.lib] : itf488cs.lib

  (or

  link

  [.obj] : dmm.obj + itf488cs.obj

  [.exe] : dmm.exe

  [.map] :

  [.lib] :

)

- Turbo C

  a. produce dmm.obj

  b. prepare project file, say, dmm.prj. → produce dmm.exe

  content of dmm.prj
  {
  dmm
  itf488cs.lib (or itf488cs.obj)
  }

File	Edit	Compiler	Project	Option	Debug
dmm			project name	dmm.prj	
itf488cs.lib(or itf488cs.obj)			break make on errors		
			clear project		

- File consist of interface functions:

itf488cs.lib	C interface functions for Small model (.lib file)
itf488cs.obj	C interface functions for Small model (.obj file)
itf488cm.lib	C interface functions for Middle model (lib file)
itf488cm.obj	C interface functions for Middle model (.obj file)
itf488cc.lib	C interface functions for Compact model (lib file)
itf488cc.obj	C interface functions for Compact model (.obj file)
itf488cl.lib	C interface functions for Large model (lib file)
itf488cl.obj	C interface functions for Large model (.obj file)
itf488cm.lib	C interface functions for Middle model (lib file)

NOTE *Turbo C and Microsoft C are using the same interface functions. You can build your own .lib file with the desired .obj file by using LIB.EXE or MASM.*

Summary of Interface Functions

Mnemonic	Description
ibinit	Initialize the AX5488 driver function routine
ibwrt	Write data string to device
ibrd	Read data string from device
ibifc	Interface clear to reset GPIB bus
ibeol	Establish an End of Line string for I/O of a device
ibdclr	Clear specified Device
ibllo	Local lockout to disable device front panel operation
iblocal	Make device go to local
ibppl	Parallel polling
ibpplc	Parallel polling configure
ibpplu	Parallel polling unconfigure
ibren	Set device to remote mode by setting Remote Enable line
ibcmd	Send command string to device
ibspl	Serial polling and return device's status
ibsts	Status of the GPIB bus
ibtout	Setup time out interval
ibtrg	Trigger specified device
ibrdm	Read data into memory with specified segment
ibwrtm	Write data from memory to device
ibxchg	Replace DOS device with GPIB device

Interface Functions

(1) Function ibinit : Initialization

- Introduction

  1. Setup base address.
  2. Setup parameter to make GPIB controller work properly.
  3. Setup interrupt level and DMA channel.
  4. Automatically check firmware segment.

  This function needed to be executed only once, must be performed before any other functions.

- Format

  Segment = ibinit (ioport, gpibaddr, config);

- Parameter

  - Entry

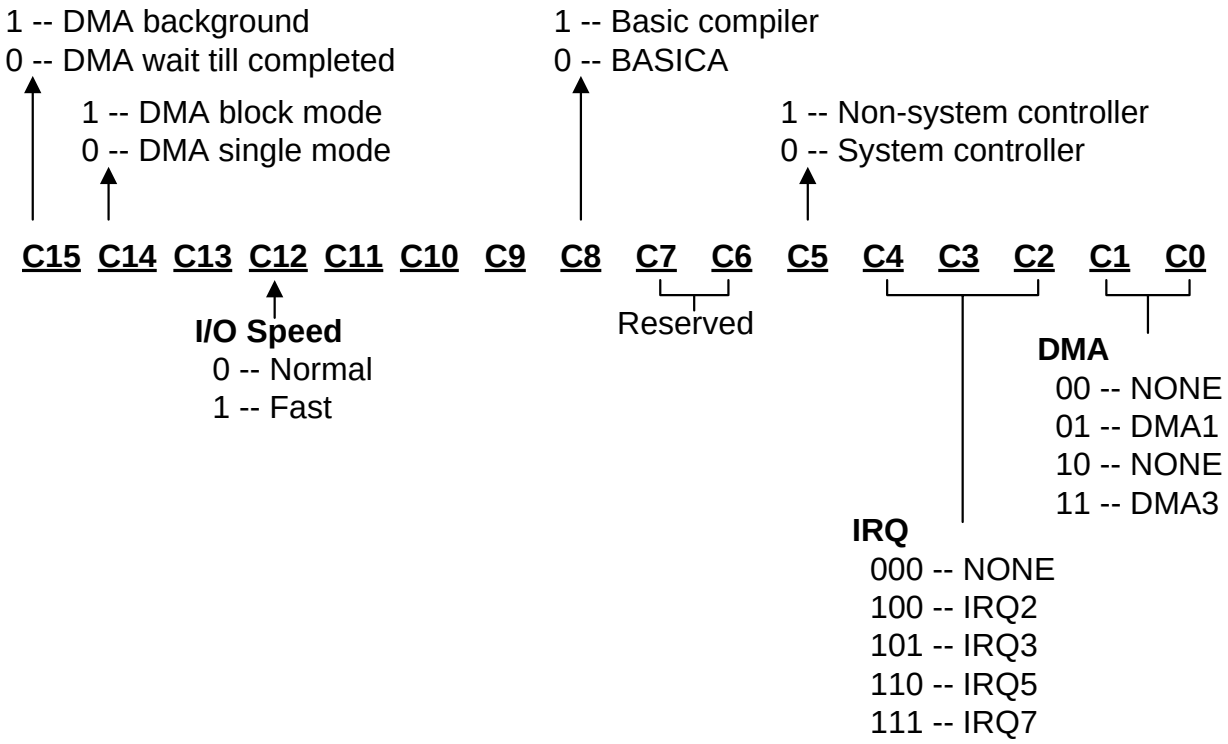
    ioport – I/O port of GPIB controller NEC7210, default 2E1 hex..

    gpibaddr – GPIB address of device, 0 through 30, default 21.

    config – Configure of the firmware routine, depending on setting bit of 16, C15…C0.

  - Return

    Segment – Address of firmware segment. Firmware does not exist if segment = 0.



The following setting are for Basic language only.

C9 : 0 – None
1 – When SRQ happen, ON KEY 20 active

C10 : 0 – None
1 – When Timeout happen, ON KEY 19 active

C11 : 0 – None
1 – When Error happen, ON ERROR active

C13 : Determine whether it is in Basic interpreter or compiler for setting of C9 through C11.
0 – for Basic interpreter
1 – for Basic compiler

- GPIB bus operation

None.

- Example

:

Set AX5488 operation mode as follows:

Basic interpreter, System controller, DMA channel 3 & single transfer mode, GPIB address 21, I/O port hex. 2E1.

```
int segment, gpibaddr, config, ioport;

   gpibaddr = 21;              /* gpib address = 21 */
   ioport = 0x2E1;             /* I/O port = 2E1 hex. */
   config = 0x0003;            /* bit 0, 1 are all 1. */
   segment = ibinit(iport, gpibaddr, config);
   if (segment==0)   {
      printf ("AX5488 not found");
      exit(0);
   }
   else
      printf ("Firmware segment = %x", segment);
```

(2) Function ibwrt : Write data string to device

- Introduction

  The routine writes a string followed a terminator to specified device.

- Format

  flag = ibwrt (gpibaddr, wrtstr, wrtstr_len);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  wrtstr – Output data string.

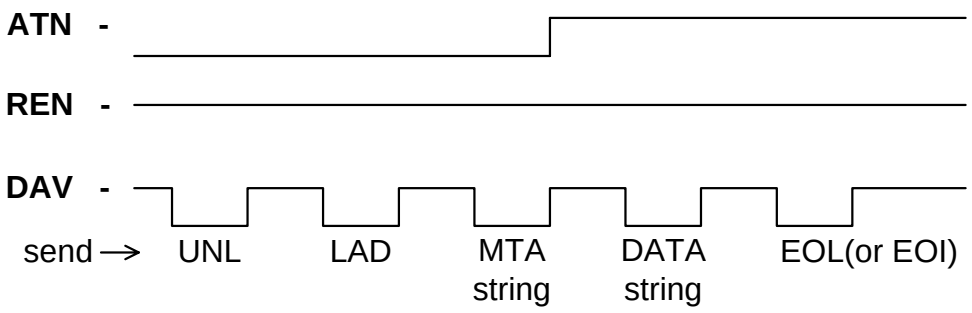
  wrtstr_len – Length of wrtstr string.

  ✧ Return

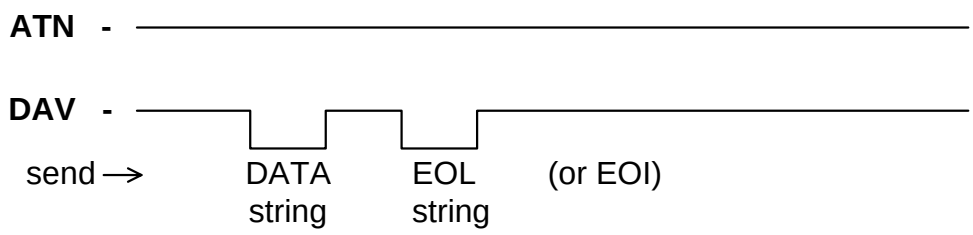
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error

- GPIB bus operation : Low – true, High -- false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

  :

  Operation :

  Write a string to Hp3478, address 23.

```
char      wrtstr[20];
int       gpibaddr, flag;
gpibaddr = 23;                  /* GPIB address = 23 */
wrtstr = "F1T3RAN3";            /* Set up the HP3478 */
flag = ibwrt(gpibaddr, wrtstr, strlen(wrtstr));
if (flag!=0)
printf ("call function failed flag = %d", flag);
```

(3) Function ibrd : Read data string from device

- Introduction

  The routine reads a string followed a terminator to specified device.

- Format

  flag = ibrd (gpibaddr, rdstr, rdstr_len);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  rdstr – Input data string.

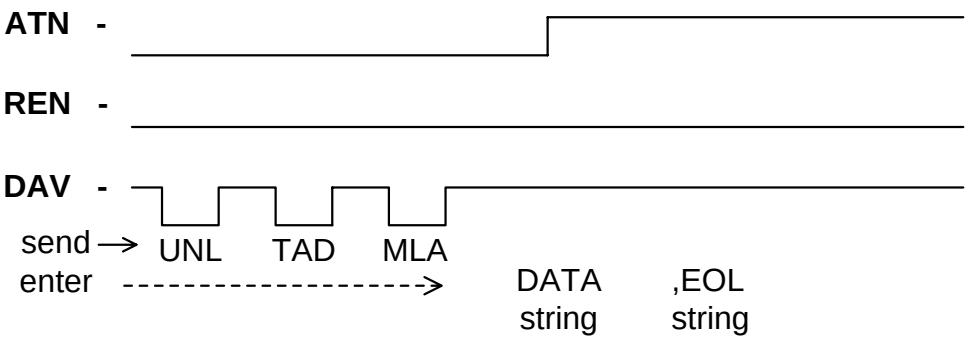
  rdstr_len – Length of rdstr string.

  ✧ Return

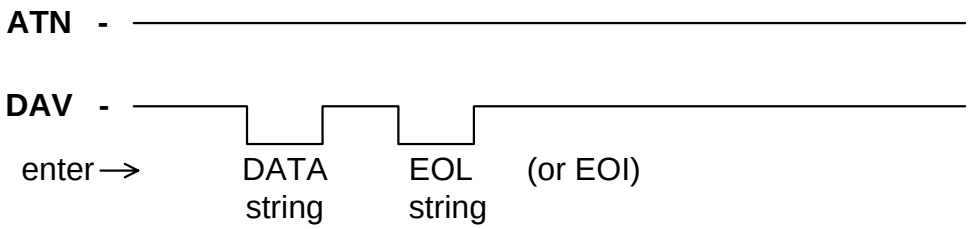
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low -- true, High -- false

  ✧ Legal GPIB address



✧ Illegal GPIB address



- Example

:

Operation :

Read a string from Hp3478, address 23.

```
int             flag;
char            *rdstr;

gpibaddr = 23;                  /* gpib address = 23 */
rdstr = "               ";
flag = ibrd (gpibaddr, rdstr, strlen(rdstr));
if (flag! = 0)
   printf ("call function failed, flag = %d", flag);
```

(4) Function ibifc : Interface clear to clear GPIB bus

- Introduction

  This routine, sending a IFC signal to clear GPIB interface bus, can be called by system controller only.

- Format

  flag = ibifc();

- Parameter

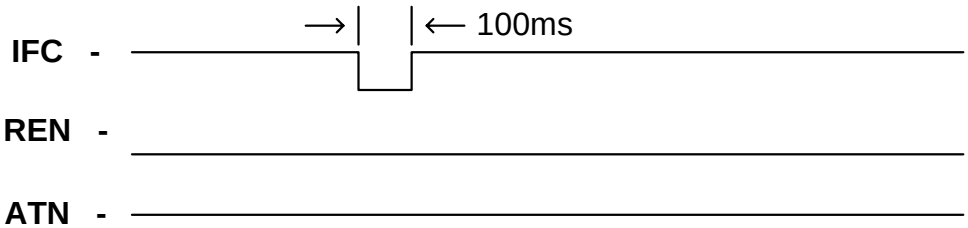
  ✧ Entry

  None

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error

- GPIB bus operation : Low -- true, High --false



- Example

```
   :
Operation :
Clear all device

int             flag;

flag = ibifc();
if (flag! = 0)
   printf ("call function failed, flag = %d", flag);
```

(5) Function ibeol : Establish an End Of Line string for I/O of a device

- Introduction

  Use this routine to establish an end of line terminator of output and input strings for specified device, otherwise the default terminators are set.
  Output terminator is default CR (13), LF (10).
  Input terminator is default LF (10).

- Format

  flag = ibeol (gpibaddr, omode, oeol, oeol_len, imode, ieol);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  omode – Output terminator mode (0-2), default 0.
  0 : Terminated when encounter OEOL$ or EOI is active.
  1 : Terminated when EOI is active.
  2 : Terminated when encounter OEOL$.

  oeol – Output End Of Line string following output string can be set up to 8 characters, default CR (13) and LF (10).

  oeol_len – Length of oeol string.

  imode – Input terminator mode (0-1), default 0.
  0 : Terminated when encounter IEOL% or EOI is active or input string full.
  1 : Terminated when EOI is true or input string full.

  ieol – Input End Of Line character, default LF (10).

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation

  NONE

- Example

  :

  Operation :

  Set hp3478 terminator, addressed 23.

```
int omode, imode, gpibaddr, flag;
char oeol[3], ieol;
gpibaddr = 23;              /* gpib address = 23 */
omode = 0;
oeol[0] = 13;
oeol[1] = 10;
oeol[2] = 0;
imode = 0;
ieol = 10;
flag = ibeol (gpibaddr, omode, oeol, strlen(oeol), imode, ieol);
if (flag!=0)
   printf ("call function failed, flag = %d", flag);
```

(6) Function ibdclr : Clear device

- Introduction

  Send SDC command to clear specified device or send DCL command to clear all devices.

- Format

  flag = ibdclr (gpibaddr);

- Parameter

  - Entry

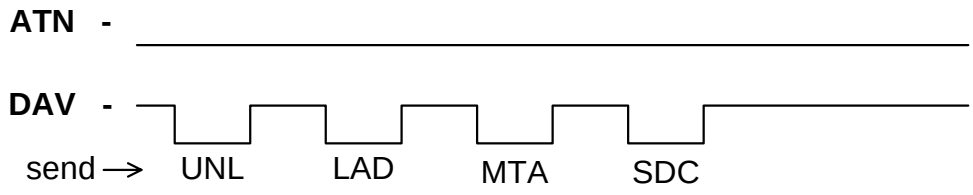
    gpibaddr – GPIB address of device, 0 through 30.

  - Return

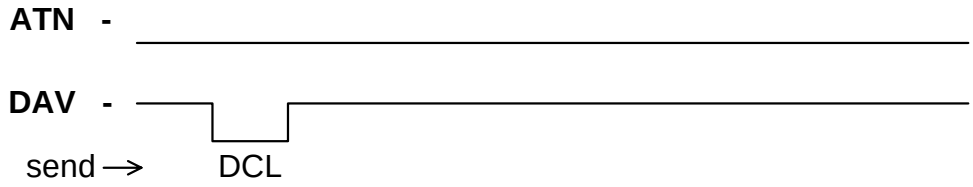
    flag – 0 – No error.
    1 – Handshake timeout.
    2 – Interface error.
    3 – Non-system controller call IBIFC.
    4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  - Legal GPIB address

  - Illegal GPIB address

- Example

:

Operation :

Clear hp3478, addressed 23.

```
int         flag, gpibaddr;

gpibaddr = 23;              /* gpib address = 23 */
flag = ibdclr (gpibaddr);
if (flag!=0)
   printf ("call function failed, flag = %d", flag);
```

(7) Function ibllo : Local Lock out to disable front panel of all devices

- Introduction

  Send LLO command to disable front panel of all devices.

- Format

  flag = ibllo();

- Parameter

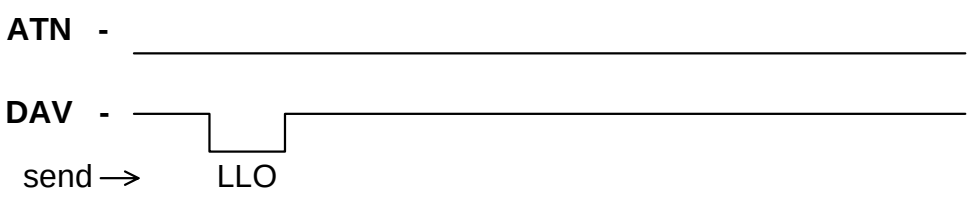
  ✧ Entry

  None

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false



- Example

```
  :
Operation :
Disable front panel of all devices

int         flag;

flag = ibllo();
if (flag!=0)
   printf ("call function failed, flag = %d", flag);
```

(8) Function iblocal : Make device goto local

- Introduction

  Sent Go To Local (GTL) command or set REN false to enable front panel of specified device.

- Format

  flag = iblocal(gpibaddr);

- Parameter

  ✧ Entry

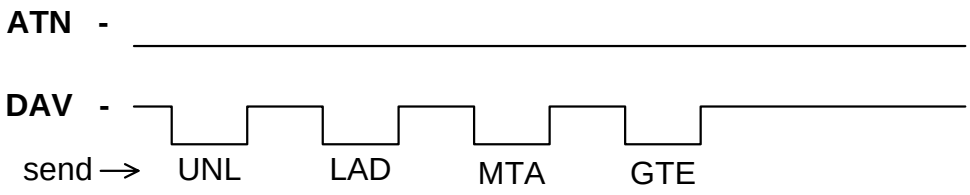
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

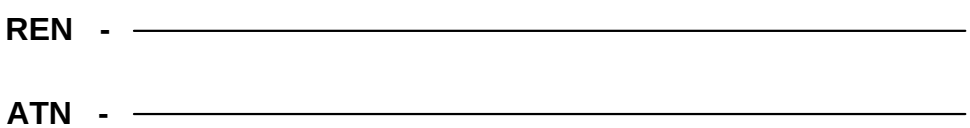
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address

  ✧ Illegal GPIB address

- Example

```
    :
Operation :
Enable hp3478's front panel.

int    gpibaddr,    flag;

gpibaddr = 23;                /* gpib address = 23 */
flag = iblocal (gpibaddr);
if (flag!=0)
    printf ("call function failed, flag = %d", flag);
```

(9) Function ibppl : Parallel Polling

- Introduction

  The Parallel polling routine allows system controller to check up to 8 devices simultaneously with SRQCHK. If any device requests service the associated bit of DIO bus is setup. After calling this routine, the DIO bus data is then transferred to variable srqchk.

  Refer to next routine IBPPLC for DIO bit definition.

- Format

  srqchk = ibppl();

- Parameter

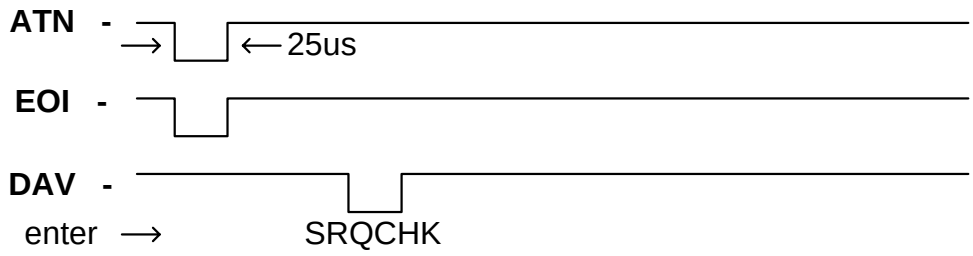
  ✧ Entry

  None

  ✧ Return

  srqchk – Return DIO bus data to SRQCHK integer after parallel polling. Only low byte makes sense.

- GPIB bus operation : Low – true, High – false



- Example

```
  :
int   srqchk;

srqchk = ibppl();
printf ("parallel polling status = %x", srqchk);
```

(10) Function ibpplc : Parallel Polling Configure

- Introduction

  The Parallel polling Configure routine allows device to specify a DIO bus line to respond to parallel poll. It uses integer CONFIG bit 0 through 2 to select DIO bus line, bit 3 to define the response state of that line.

- Format

  flag = ibpplc(gpibaddr, config);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  config – An integer to configure a device to respond to parallel polling. Only the low nibble makes sense.

  BIT 3 2 1 0

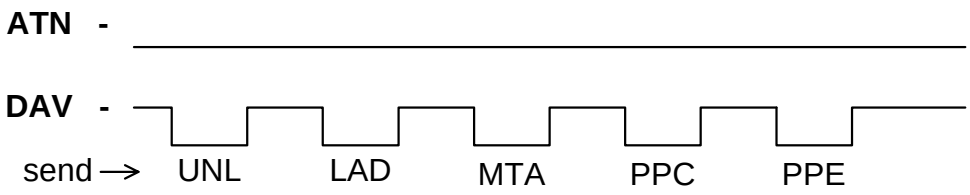
  000 – DIO1
  001 – DIO2
  :
  007 – DIO8

  BIT 3 – Determine the state of data line(DIO#)
  0 : low state
  1 : high state

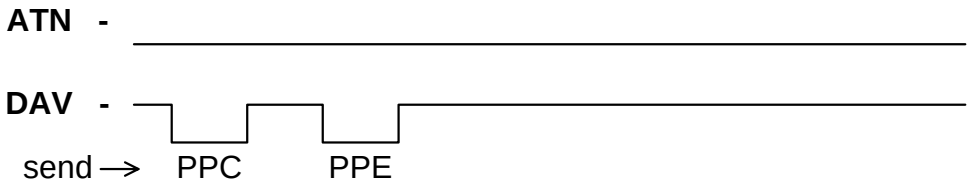
  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error

- GPIB bus operation : Low – true, High – false
  - ✧ Legal GPIB address



  - ✧ Illegal GPIB address



- Example

:

Operation :

Configure hp3478, addressed 23.

```
int config,      gpibaddr,      flag;

config = 0x00                   /* select DIO0, reflect with low state */
gpibaddr = 23;                  /* gpib address = 23 */
flag = ibpplc (gpibaddr, config);
if (flag!=0)
    printf ("call function failed, flag = %d", flag);
```

(11) Function ibpplu : Parallel Polling Unconfigure

- Introduction

  The Parallel polling Unconfigure routine is used to tell specified device or all devices on the interface bus not to respond to the parallel polling. If GPIBADDR is legal, the addressed device will be unconfigured, otherwise all devices will be unconfigured.

- Format

  flag = ibpplu(gpibaddr);

- Parameter

  - Entry

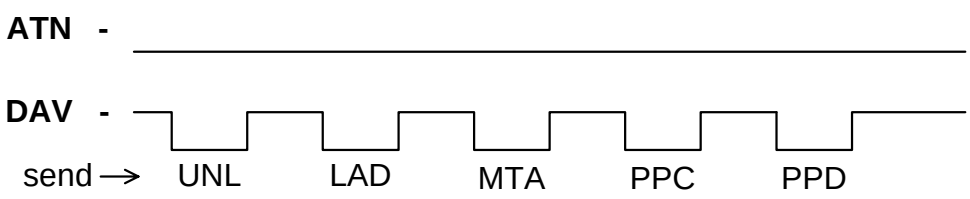
    gpibaddr – GPIB address of device, 0 through 30.

  - Return

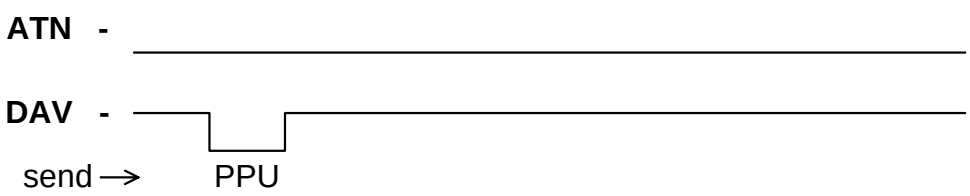
    flag – 0 – No error.
    1 – Handshake timeout.
    2 – Interface error.
    3 – Non-system controller call IBIFC.
    4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  - Legal GPIB address



  - Illegal GPIB address



- Example

:

Operation :

Unconfigure hp3478, addressed 23.

```
int    gpibaddr,      flag;

gpibaddr = 23;                /* gpib address = 23 */
flag = ibpplu (gpibaddr);
if (flag!=0)
   printf ("call function failed, flag = %d", flag);
```

(12) Function ibren : Remote Enable

- Introduction

  This routine is used set specified device or all devices on the interface bus to remote mode. If GPIBADDR is legal, the addressed device will be in remote mode, otherwise all devices will be in remote mode.

- Format

  flag = ibren(gpibaddr);

- Parameter

  ✧ Entry

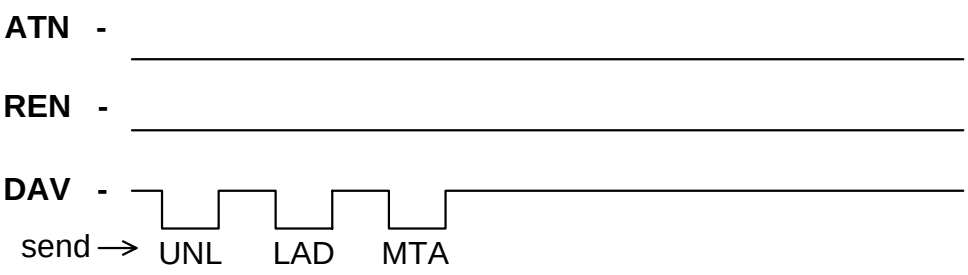
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

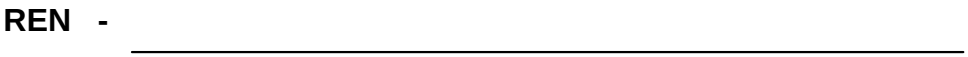
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address

  ✧ Illegal GPIB address

- Example

:

Operation :

Set hp3478, addressed 23 to remote mode.

```
gpibaddr = 23;              /* gpib address = 23 */
flag = ibren (gpibaddr);
if (flag!=0)
   printf ("call function failed, flag = %d", flag);
```

(13) Function ibcmd : Send command code

- Introduction

  To send command directly to gpib interface bus, or to control some devices simultaneously, use this routine.

- Format

  flag = ibcmd(cmdstr, cmdstr_len);

- Parameter

  ✧ Entry

  cmdstr – GPIB bus command string sent to GPIB bus. There are two group commands, one is bus command sending character to the bus, the other is action command which require some actions from GPIB controller.

  cmdstr_len – Length of cmdstr string.

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error

  ✧ Bus commands are listed as follows:

Mnemonic	Hex.	Mode	Description
GTL	01	Address	Goto Local. Set addressed device to local mode.
SDC	04	Address	Selected Device Clear. Reset addressed device.
PPC	05	Address	Parallel Poll Configure.
GET	08	Address	Group Execute Trigger. Trigger all listeners on the bus.
TCT	09	Address	Take Control. Transfer control to another device.
LLO	11	Global	Local Lockout to disable front panel of all device.

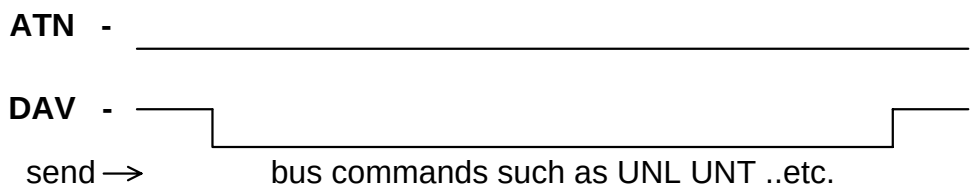
Mnemonic	Hex.	Mode	Description
DCL	14	Global	Device Clear
PPU	15	Global	Parallel Poll Unconfigure.
SPE	18	Global	Serial Poll Enable.
SPD	19	Global	Serial Poll Disable.
REN	1F	Global	Remote Enable.
UNL	3F	Address	Unlisted. Unaddress the listener.
UNT	5F	Address	Untalk. Unaddress the talker.
MLA	35	Address	My Listen address.
MTA	55	Address	My talk address.
PPD	70	Global	Parallel poll disable.

REMARK *Global means that the commands are sent to all devices on the bus. Address means that the commands are sent to address device only.*

✧ Action commands are listed as follows:

Mnemonic	Description
LISTEN	Values following this word are listener addresses.
TALK	Value following this word is talker address.
SEC	Values following this word are secondary commands
PPE	Values following this word are parallel poll configure.
DATA	Set ATN false for data mode.
CMD	Set ATN true for command mode.
IFC	Send IFC pulse for 100ms.
EOI	Set EOI true after the last byte of output string is sent.

- GPIB bus operation : Low – true, High – false



- Example

:

Operation :

Send bus commands to trigger hp3478, addressed 23.

```
int             flag;
char            *cmdstr;

cmdstr = "UNT UNL MTA LISTEN 23 GET";
flag = ibcmd (cmdstr, strlen (cmdstr));
if (flag!=0)
   printf ("call function failed, flag = %d", flag);
```

(14) Function ibspl : Serial Polling

- Introduction

  The Serial polling routine allow system controller to know whether the specified device request service by checking the returned variable srqchk low byte.

- Format

  srqchk = ibspl (gpibaddr);

- Parameter

  ✧ Entry

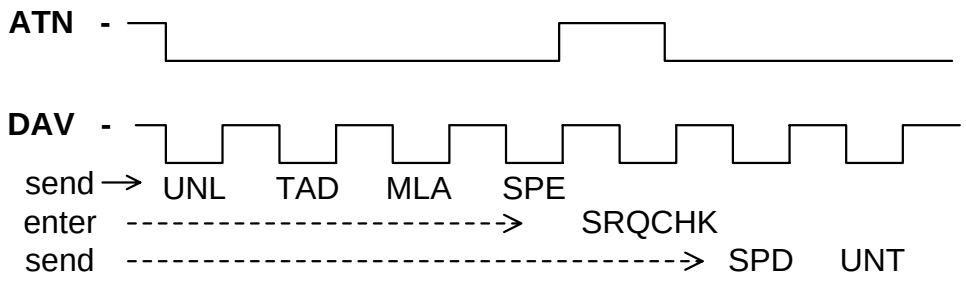
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

  srqchk – Return DIO bus data to SRQCHK integer after serial polling. Only low byte makes sense.

  BIT 7 6 5 4 3 2 1 0
  |
  SRQ
  0 : Active
  1 : None
  Bit 7 : reserved
  Bit 1 through 5 are defined depending on device.

- GPIB bus operation : Low – true, High – false



- Example

:

Operation :

Serial poll hp3478, addressed 23 after triggered to determine whether the data is ready to be read by checking return variable SRQCHK bit 0.

```
int    gpibaddr,      srqchk;

gpibaddr = 23;                  /* HP3478 addressed = 23 */
srqchk = ibspl (gpibaddr);
printf ("serial polling status = %x", srqchk);
```

(15) Function ibsts : Status of GPIB bus

- Introduction

  This routine allows you to monitor the status of GPIB bus, NEC7210 controller and etc. by checking the returned variable STATUS.

- Format

  status = ibsts (cfg);

- Parameter

  ✧ **Entry**

  cfg – A 16 bit configure variable to determine what status to be returned.

0 – 7	:	Return NEC7210 read register 0 – 7.
8	:	Return error number of last call routine.
9	:	Return length of output or input string.
10	:	Return Timeout interval (ms).
11	:	Return I/O port of NEC7210.
12	:	Return DMA and IRQ setting.
13 – 15	:	None.

  REMARK *If bit 8 is set, then error NO. is returned as follows.*

Error No.	Description
0	No error
1	Handshake timeout
2	Interface error
3	Non-system controller call ibifc
4	Parameter setting error

  ✧ **Return**

  status – Returned status variable depending on CFG variable setting.

- GPIB bus operation :

  None.

- Example

  :

  Operation :

  Read error number of last call routine.

```
int    cfg, status;

cfg = 0x10;              /* return error NO. of last */
status = ibsts (cfg);
printf ("status = %x", status);
```

(16) Function ibtout : Setup Timeout interval

- Introduction

  This routine allows you to set timeout interval. If there is any malfunction in handshake, then the interface line is stuck until timeout interval is up.

- Format

  flag = ibtout (intvl);

- Parameter

  ✧ Entry

  intvl – Timeout period (1 – 65535).
  0 : Disable timeout routine.
  N : N units N = 1 to 32767
  N : N + 65535 units N = -1 to -32767

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation

  None

- Example

  :

  Operation :

  Set timeout interval for gpib interface bus.

```
int   intvl,      flag;

intvl = 100;              /* timeout period 100 units */
flag = ibtout (intvl);
if (flag! = 0)
   printf ("call function failed, flag = %d", flag);
```

(17) Function ibtrg : Trigger device

- Introduction

  This routine is used to trigger specified device or all devices on the interface bus by sending Group Execute Trigger (GTE) to the interface bus. If GPIBADDR is legal, the addressed device will be triggered, otherwise all devices will be triggered.

- Format

  flag = ibtrg (gpibaddr);

- Parameter

  ✧ Entry

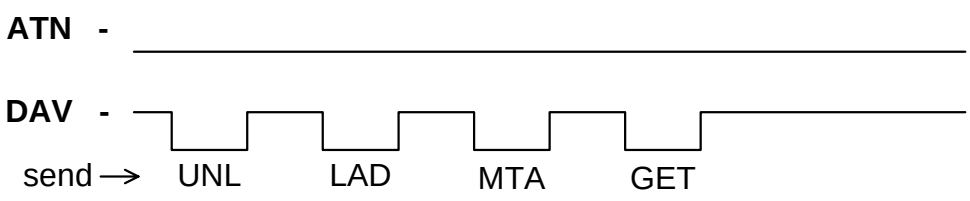
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

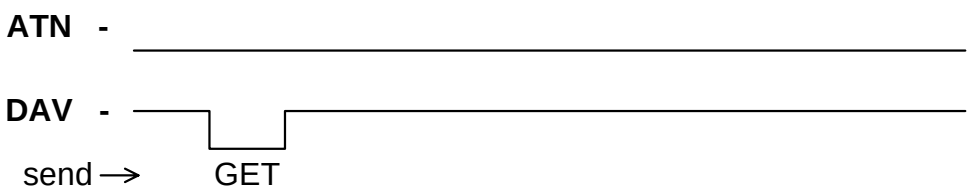
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address

  ✧ Illegal GPIB address

- Example

:

Operation :

Trigger hp3478, addressed 23.

```
int    gpibaddr,      flag;

gpibaddr = 23;              /* gpib address = 23 */
flag = ibtrg (gpibaddr);
if (flag != 0)
   printf ("call function failed, flag = %d", flag);
```

(18) Function ibrdm : Read data string from device to Memory

- Introduction

  This routine enables you to read a long data string up to 64K bytes from a device or interface bus into memory buffer with specified segment. Data string will be always placed from offset 0.

  The following condition will terminate the reading :

  - Reading number is reached.
  - Terminator or EOI is encountered.
  - Timeout.

- Format

  flag = ibrdm (gpibaddr, mseg, len);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  mseg – Segment of specified memory buffer. Be sure that the segment is not in use by your program.
  The offset start from 0 to 65535.

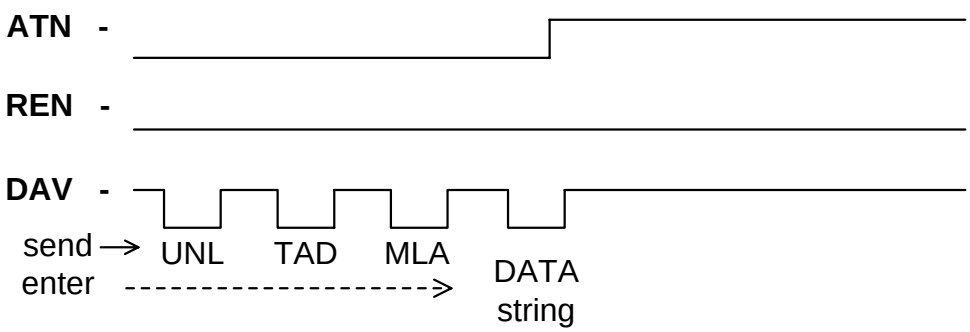
  len – Length of string 0 through 65535.

  ✧ Return

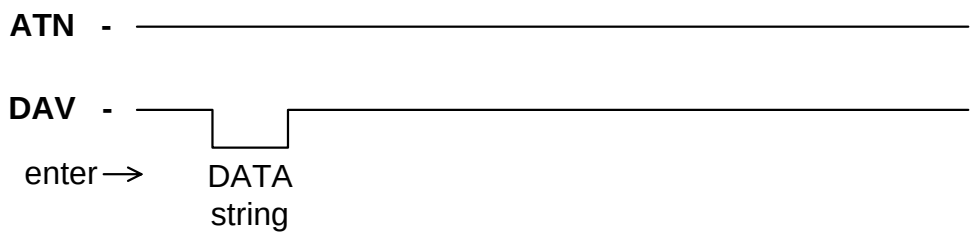
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



✧ Illegal GPIB address



- Example

:

Operation :

Read lots of bytes of string from Hp3478, addressed 23.

```
unsigned int        rdno,i;
char far            *rdbuf;
char                tmp[2];

gpibaddr = 23;                          /* gpib address = 23 */
mseg = 0x6000;                          /* memory segment = 0x6000 */
len = 1024;                             /* collect 1024 bytes data */
flag = ibrdm (gpibaddr, mseg, len);     /* collect data now */
cfg = 9;                                /* Set configure = 9 for reading number */
rdno = 0;                               /* Reset reading number */
rdno = ibsts (cfg);
rdbuf = (char far*) 0x60000000L;
for    (i = 0; i < rdno; i++)    {
    sprintf (tmp, "%c", rdbuf[i]);
    printf ("%s", tmp);
}
```

(19) Function ibwrtm : Write data string from Memory to device or interface

- Introduction

  This routine enables you to write a long data string up to 64K bytes from memory buffer with specified segment to a device or interface bus. The starting offset of the data number is reached.

  The following condition will terminate the writing :
  - Writing number is reached.
  - Timeout.

- Format

  flag = ibwrtm (gpibaddr, mseg, len);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  mseg – Segment of specified memory buffer. Be sure that the segment is not in use by your program.
  The offset start from 0 to 65535.

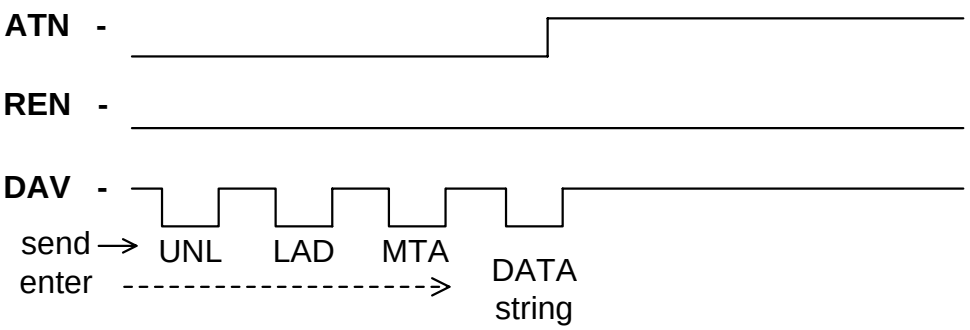
  len – Length of string 0 through 65535.

  ✧ Return

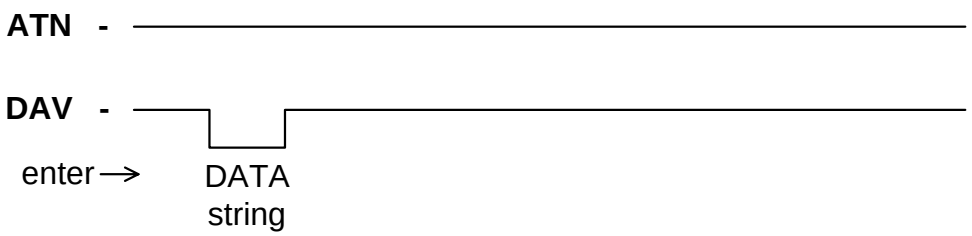
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



✧ Illegal GPIB address



- Example

:

Operation :

Write a 1024 bytes of string allocated in memory with segment 0x6000 to Hp3478, addressed 23.

```
gpibaddr = 23;                          /* gpib address = 23 */
mseg = 0x6000;                          /* memory segment = 0x6000 */
len = 1024;                             /* write 1024 bytes data */
flag = ibwrtm(gpibaddr, mseg, len);     /* Write data now */
if (flag != 0)
   printf ("call function failed, flag = %d", flag);
```

(20) Function ibxchg : Replace DOS device with GPIB device

- Introduction

  This routine enables you to replace DOS device with GPIB device. Then, you can access the GPIB device using DOS commands. The DOS devices are LPT1:, LPT2:, LPT3:, COM1: and COM2:.

- Format

  flag = ibxchg (gpibaddr, unit);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  unit – Define which device is to be replaced with GPIB device.
  1 – LPT1:
  2 – LPT2:
  3 – LPT3:
  4 – COM1:
  5 – COM2:

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation

  None

- Example

  :

  Operation :

  Install GPIB printer, addressed 8 in place of DOS LPT1:

```
gpibaddr = 8;                    /* gpib address = 8 */
unit = 1;                        /* LPT1: */
flag = ibxchg (gpibaddr, unit);  /* Change now */
fprintf ("LPT1", "A");           /* output character A to gpib printer */
```

Example Program

- **DMM.C :** *Voltage Measurement with DMM (HP3478)*
- **SENDCMD.C :** *Voltage Measurement with DMM (HP3478) by sending commend string to interface bus.*
- **AD500.C :** ***R****esistance Measurement with DMM (HP3478) and AD500 programmable measurement unit.*

Programming with AX5488 Firmware Routine in Turbo Pascal

Introduction

In Basic language, it can be called directly by define segment and offset, but for other language, it cannot work by calling this way. So, we develop interface function for you to call the firmware routine. These interface functions support the following languages:

- Turbo Pascal

Link Interface Function Usage

This driver having to be linked while compiling user's programs, is just prepared for Turbo Pascal. The following examples show you how to link it in small model.

- Link driver : *itf488tp.obj and itf488tp.inc must be in working path.*

```
Program         dmm;
Uses            crt;
{ $L    itf488tp.obj }
{ $I    itf488tp.inc }
BEGIN
   { Main Program Area }
END
```

- File consist of interface functions :

itf488tp.obj AX5488 Turbo Pascal interface functions.

itf488tp.inc Interface functions declaration.

Interface Functions Declaration : ift488tp.inc

```
{   Begin   }
Type lstring = string[255];

Function    ibinit (ioport, myaddr, sys_cfg : integer) : integer; external;
Function    ibwrt (addr : integer; var    wrtstr : lstring) : integer; external;
Function    ibrd (addr:integer; var rdstr : latring):integer; external
Function    ibifc : integer; external;
Function    ibeol (addr.outeol: integer; var outeolstr:lstring; ineol, ineolb:integer) :
            integer; external;
Function    ibdclr (addr : integer) : integer; external;
Function    ibllo : integer; external;
Function    iblocal (addr : integer) : integer; external;
Function    ibppl : integer; external;
Function    ibpplc (addr, config : integer) : integer; external;
Function    ibpplu (addr: integer) : integer; external;
Function    ibren (addr : integer) : integer; external;
Function    ibcmd (var cmdstr:lstring) : integer; external;
Function    ibspl (addr : integer) : integer; external;
Function    ibsts (condition : integer) : integer; external;
Function    ibtout (timeset : integer) : integer; external;
Function    ibtrg (addr : integer) : integer; external;
Function    ibrdm (addr, dataseg, datalen : integer) : integer; external;
Function    ibwrtm (addr, dataseg, datalen : integer) : integer; external;
Function    ibxchg (addr, portno:integer) : integer; external;
{   End   }
```

Summary of Interface Functions

Mnemonic	Description
ibinit	Initialize the AX5488 driver function routine
ibwrt	Write data string to device
ibrd	Read data string from device
ibifc	Interface clear to reset GPIB bus
ibeol	Establish an End of Line string for I/O of a device
ibdclr	Clear specified Device
ibllo	Local lockout to disable device front panel operation
iblocal	Make device go to local
ibppl	Parallel polling
ibpplc	Parallel polling configure
ibrplu	Parallel polling unconfigure
ibren	Set device to remote mode by setting Remote Enable line
ibcmd	Send command string to device
ibspl	Serial polling and return device's status
ibsts	Status of the GPIB bus
ibtout	Setup time out interval
ibtrg	Trigger specified device
ibrdm	Read data into memory with specified segment
ibwrtm	Write data from memory to device
ibxchg	Replace DOS device with GPIB device

Interface Functions

(1) Function ibinit : Initialization

- Introduction

  1. Setup base address.
  2. Setup parameter to make GPIB controller work properly.
  3. Setup interrupt level and DMA channel.
  4. Automatically check firmware segment.

  This function needed to be executed only once, must be performed before any other functions.

- Format

  Segment := ibinit (ioport, gpibaddr, config);

- Parameter

  ✧ Entry

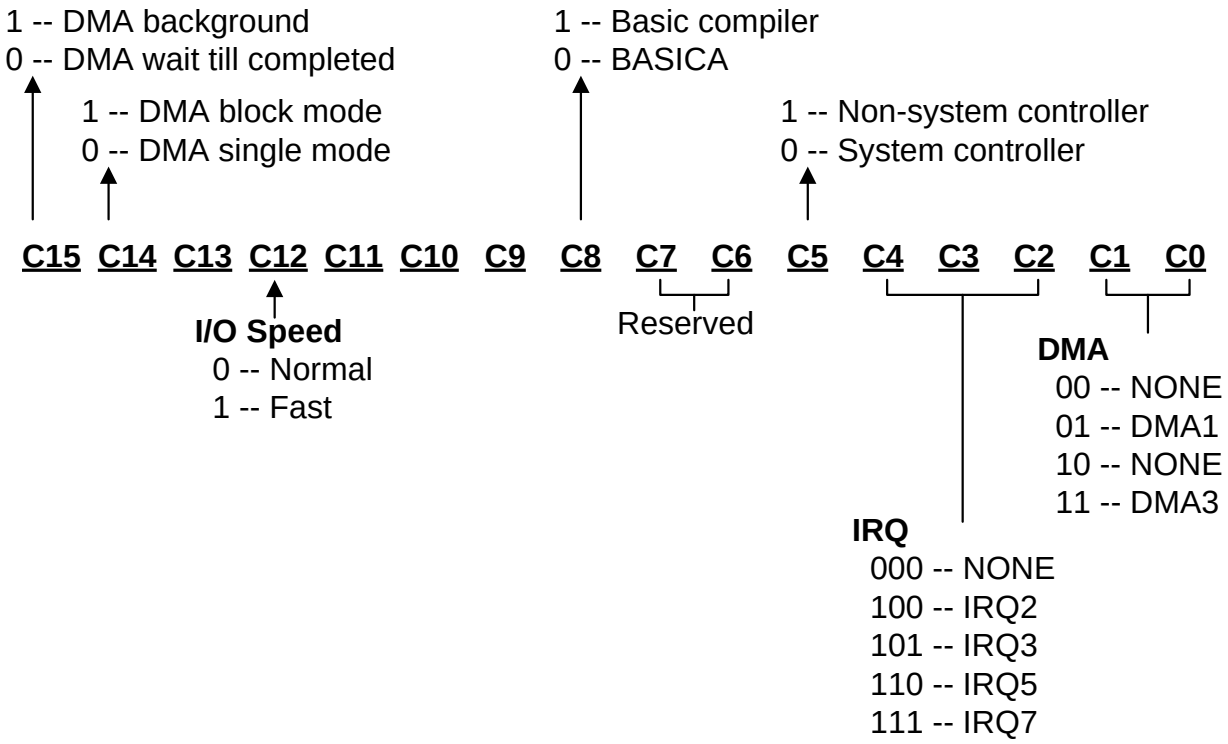
  ioport – I/O port of GPIB controller NEC7210, default 2E1 hex..

  gpibaddr – GPIB address of device, 0 through 30, default 21.

  config – Configure of the firmware routine, depending on setting bit of 16, C15…C0.

  ✧ Return

  Segment – Address of firmware segment. Firmware does not exist if segment := 0.



The following setting are for Basic language only.

C9 : 0 – None
1 – When SRQ happen, ON KEY 20 active

C10 : 0 – None
1 – When Timeout happen, ON KEY 19 active

C11 : 0 – None
1 – When Error happen, ON ERROR active

C13 : Determine whether it is in Basic interpreter or compiler for setting of C9 through C11.
0 – for Basic interpreter
1 – for Basic compiler

- GPIB bus operation

None.

- Example

:

Set AX5488 operation mode as follows:

Basic interpreter, System controller, DMA channel 3 & single transfer mode, GPIB address 21, I/O port hex. 2E1.

```
Var segment, gpibaddr, config, ioport : integer;

  gpibaddr := 21;          { gpib address := 21 }
  ioport := $2E1;          { I/O port := 2E1 hex. }
  config := $0003;         { bit 0, 1 are all 1. }
  segment := ibinit(ioport, gpibaddr, config);
  if (segment=0) then
     begin
        writeln (' AX5488 not found ');
        exit;
     }
  else
     writeln (' Firmware segment =', segment);
```

(2) Function ibwrt : Write data string to device

- Introduction

  The routine write a string followed a terminator to specified device.

- Format

  flag = ibwrt (gpibaddr, wrtstr);

- Parameter

  - Entry

    gpibaddr – GPIB address of device, 0 through 30.

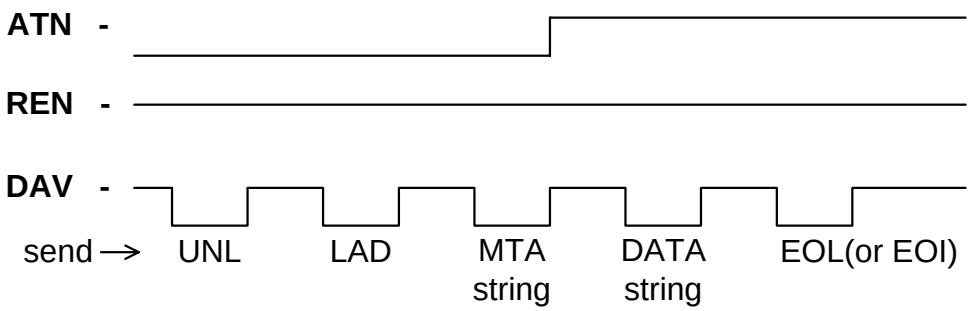
    wrtstr – Output data string.

  - Return

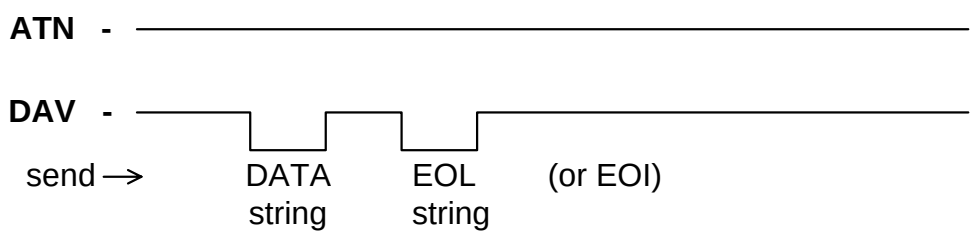
    flag – 0 – No error.
    1 – Handshake timeout.
    2 – Interface error.
    3 – Non-system controller call IBIFC.
    4 – Parameter setting error.

- GPIB bus operation : Low – true, High -- false

  - Legal GPIB address



  - Illegal GPIB address



- Example

:

Operation :
Write a string to Hp3478, address 23.

```
Var
      rdstr    :    lstring;

Var
      gpibaddr, flag : integer;
      wrtstr : lstring;              { lstring defined in itf488tp.inc }

gpibaddr := 23;                      { GPIB address := 23 }
wrtstr := ' F1T3RAN3 ';              { Set up the HP3478 }
flag := ibwrt(gpibaddr, wrtstr, strlen(wrtstr));
if (flag <> 0) then
      writeln (' call function failed, flag =' ,flag);
```

(3) Function ibrd : Read data string from device

- Introduction

  The routine read a string followed a terminator to specified device.

- Format

  flag := ibrd (gpibaddr, rdstr);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

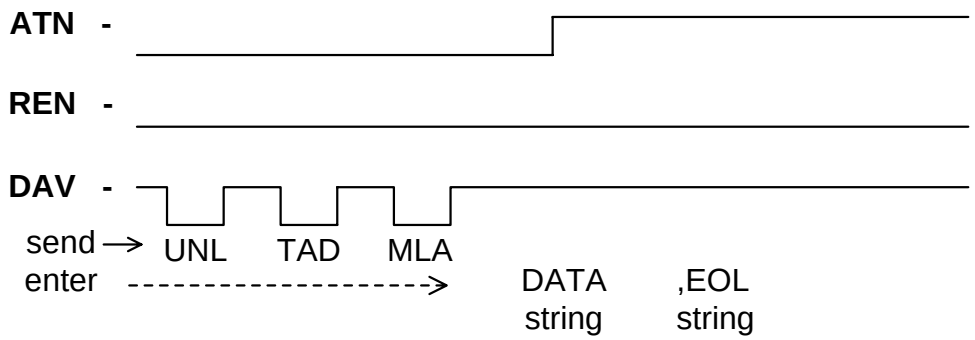
  rdstr – Input data string.

  ✧ Return

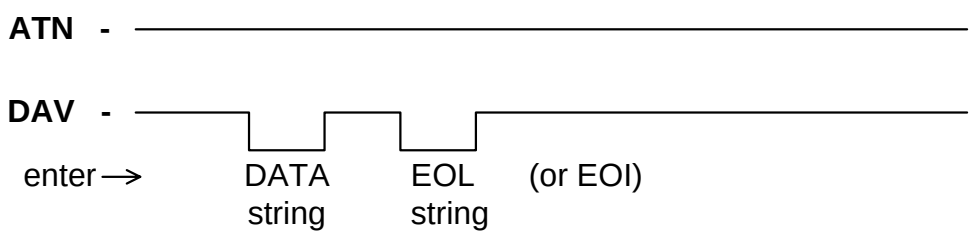
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low -- true, High -- false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

:

Operation :

Read a string from Hp3478, address 23.

```
Var
      flag      :     integer;
      rdstr     :     lstring;

gpibaddr := 23;                  { gpib address := 23 }
rdstr := '                ';
flag := ibrd(gpibaddr, rdstr);
if (flag <> 0) then
      writeln (' call function failed, flag =' ,flag);
```

(4) Function ibifc : Interface clear to clear GPIB bus

- Introduction

  This routine, sending a IFC signal to clear GPIB interface bus, can be called by system controller only.

- Format

  flag := ibifc;

- Parameter

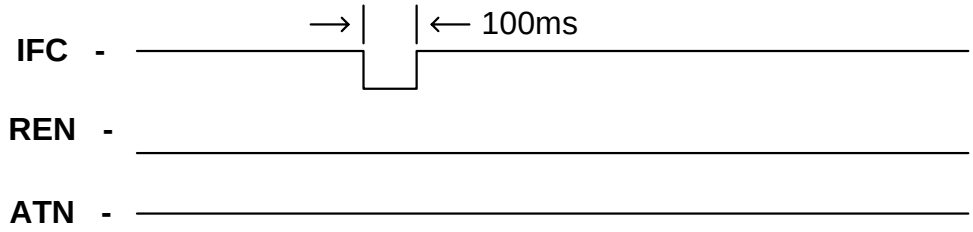
  ✧ Entry

  None

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low -- true, High -- false



- Example

```
   :
Operation :
Clear all device

Var
      flag   :   integer;

flag := ibifc;
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(5) Function ibeol : Establish an End Of Line string for I/O of a device

- Introduction

  Use this routine to establish an end of line terminator of output and input strings for specified device, otherwise the default terminators are set.
  Output terminator is default CR (13), LF (10).
  Input terminator is default LF (10).

- Format

  flag := ibeol (gpibaddr, omode, oeol, imode, ieol);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  omode – Output terminator mode (0-2), default 0.
  0 : Terminated when encounter OEOL$ or EOI is active.
  1 : Terminated when EOI is active.
  2 : Terminated when encounter OEOL$.

  oeol – Output End Of Line string following output string can be set up to 8 characters, default CR (13) and LF (10).

  imode – Input terminator mode (0-1), default 0.
  0 : Terminated when encounter IEOL% or EOI is active or input string full.
  1 : Terminated when EOI is true or input string full.

  ieol – Input End Of Line character, default LF (10).

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation

  NONE

- Example

:

Operation :

Set hp3478 terminator, addressed 23.

```
Var
        omode, imode, gpibaddr, flag : integer;
        oeol    :    lstring;

gpibaddr := 23;                    { gpib address := 23 }
omode := 0;
oeol := chr(13);
imode := 0;
flag := ibeol(gpibaddr, omode, oeol, imode, 10);
if (flag <> 0) then
        writeln (' call function failed, flag =' ,flag);
```

(6) Function ibdclr : Clear device

- Introduction

  Send SDC command to clear specified device or send DCL command to clear all devices.

- Format

  flag := ibdclr (gpibaddr);

- Parameter

  ✧ Entry

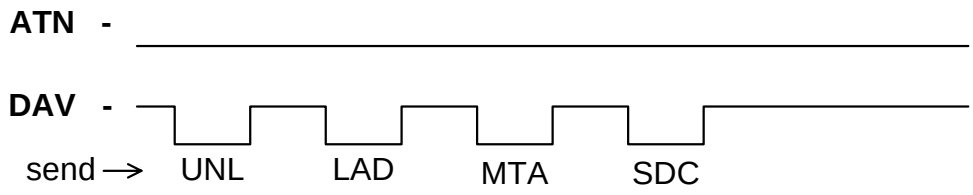
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

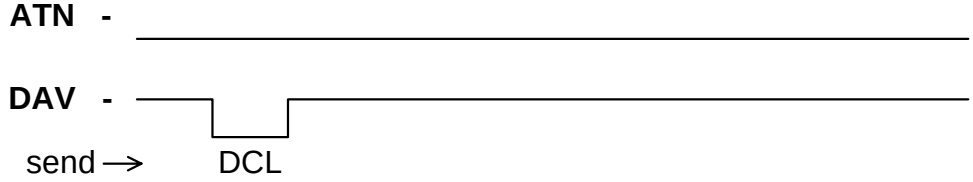
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address

  ✧ Illegal GPIB address

- Example

```
    :
Operation :
Clear hp3478, addressed 23.

Var
      flag,   gpibaddr : integer;

gpibaddr := 23;              { gpib address := 23 }
flag := ibdclr (gpibaddr);
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(7) Function ibllo : Local Lock out to disable front panel of all devices

- Introduction

  Send LLO command to disable front panel of all devices.

- Format

  flag := ibllo;

- Parameter

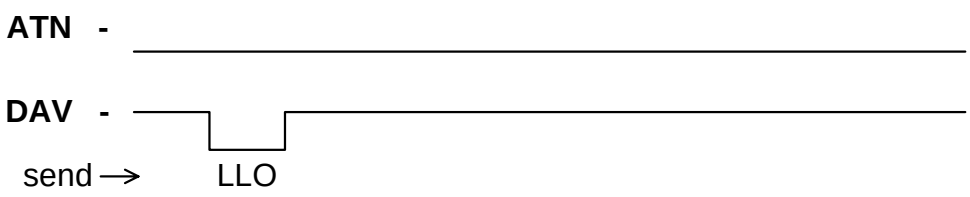
  ✧ Entry

  None

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false



- Example

```
  :
Operation :
Disable front panel of all devices

Var
      flag   :   integer;

flag := ibllo;
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(8) Function iblocal : Make device goto local

- Introduction

  Sent Go To Local (GTL) command or set REN false to enable front panel of specified device.

- Format

  flag := iblocal(gpibaddr);

- Parameter

  ✧ Entry

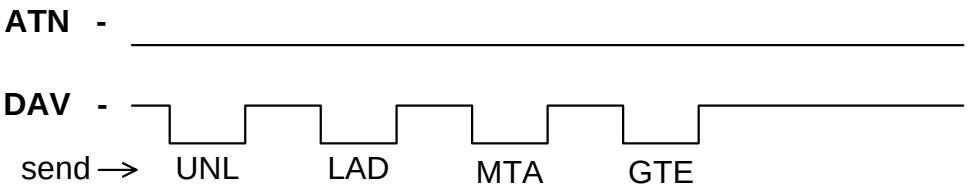
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

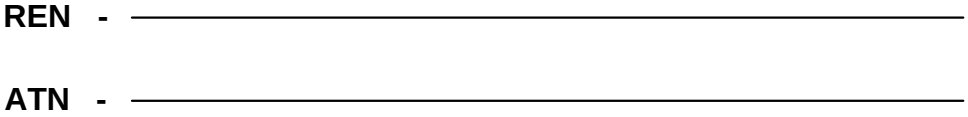
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address

  ✧ Illegal GPIB address

- Example

```
    :
Operation :
Enable hp3478's front panel

Var
      gpibaddr,   flag   :   integer;

gpibaddr := 23;              { gpib address := 23 }
flag := iblocal (gpibaddr);
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(9) Function ibppl : Parallel Polling

- Introduction

  The Parallel polling routine allows system controller to check up to 8 devices simultaneously with SRQCHK. If any device requests service the associated bit of DIO bus is setup. After calling this routine, the DIO bus data is then transferred to variable srqchk.

  Refer to next routine IBPPLC for DIO bit definition.

- Format

  srqchk := ibppl;

- Parameter

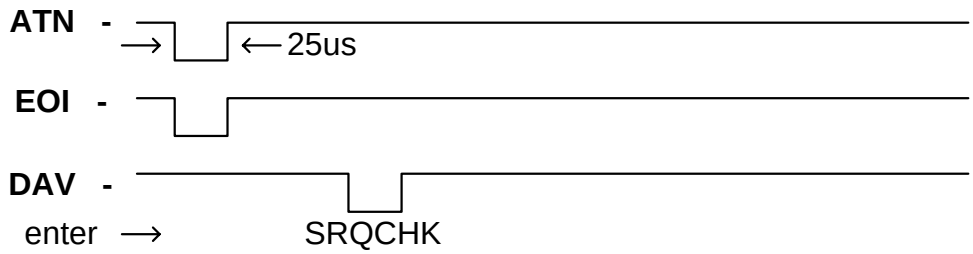
  ✧ Entry

  None

  ✧ Return

  srqchk – Return DIO bus data to SRQCHK integer after parallel polling. Only low byte makes sense.

- GPIB bus operation : Low – true, High – false



- Example

```
  :
Var
    srqchk   :   integer;

srqchk := ibppl;
writeln (' parallel polling status =', srqchk);
```

(10) Function ibpplc : Parallel Polling Configure

- Introduction

  The Parallel polling Configure routine allow device to specify a DIO bus line to respond to parallel poll. It uses integer CONFIG bit 0 through 2 to select DIO bus line, bit 3 to define the response state of that line.

- Format

  flag := ibpplc(gpibaddr, config);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  config – An integer to configure a device to respond to parallel polling. Only the low nibble makes sense.

  BIT 3 2 1 0

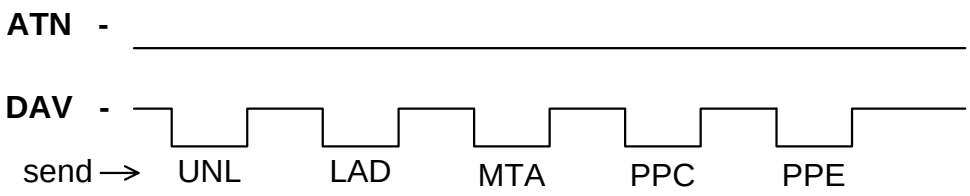
  000 – DIO1
  001 – DIO2
  :
  007 – DIO8

  BIT 3 – Determine the state of data line(DIO#)
  0 : low state
  1 : high state

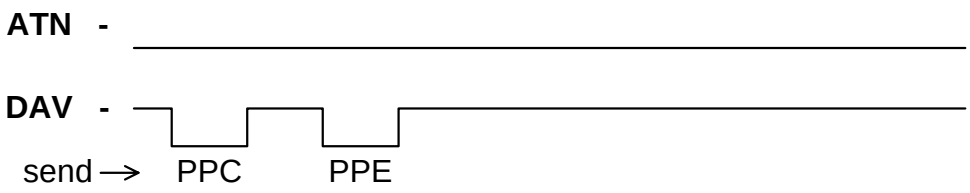
  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error

- GPIB bus operation : Low – true, High – false
  - ✧ **Legal GPIB address**



  - ✧ **Illegal GPIB address**



- Example

:

Operation :

Configure hp3478, addressed 23.

```
Var
      config,   gpibaddr,   flag   :   integer;

config := $00;              { select DIO0, reflect with low state }
gpibaddr := 23;             { gpib address := 23 }
flag := ibpplc (gpibaddr, config);
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(11) Function ibpplu : Parallel Polling Unconfigure

- Introduction

  The Parallel polling Unconfigure routine is used to tell specified device or all devices on the interface bus not to respond to the parallel polling. If GPIBADDR is legal, the addressed device will be unconfigured, otherwise all devices will be unconfigured.

- Format

  flag := ibpplu(gpibaddr);

- Parameter

  ✧ Entry

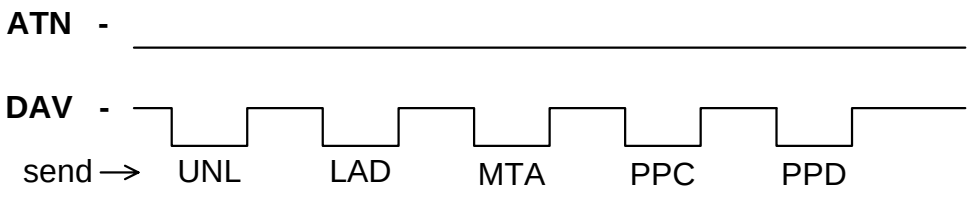
  Gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

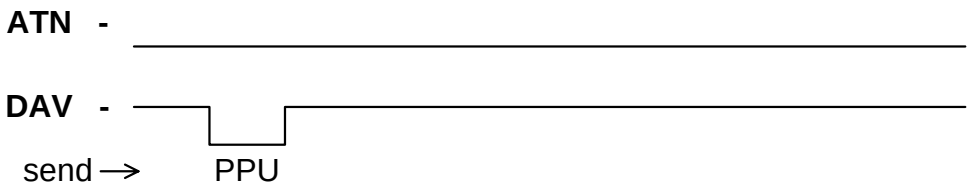
  Flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

```
    :
Operation :
Unconfigure hp3478, addressed 23.

Var
      gpibaddr,   flag   :   integer;

gpibaddr := 23;              { gpib address := 23 }
flag := ibpplu (gpibaddr);
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(12) Function ibren : Remote Enable

- Introduction

  This routine is used set specified device or all devices on the interface bus to remote mode. If GPIBADDR is legal, the addressed device will be in remote mode, otherwise all devices will be in remote mode.

- Format

  flag := ibren(gpibaddr);

- Parameter

  ✧ Entry

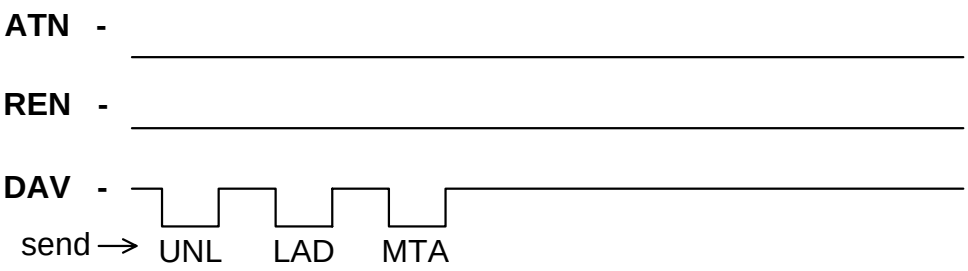
  Gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

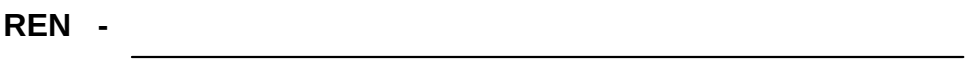
  Flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address

  ✧ Illegal GPIB address

- Example

```
    :
Operation :
Set hp3478, addressed 23 to remote mode.

Var
      gpibaddr,   flag   :   integer;

gpibaddr := 23;              { gpib address := 23 }
flag := ibren (gpibaddr);
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(13) Function ibcmd : Send command code

- Introduction

  To send command direct to gpib interface bus, or to control some devices simultaneously, use this routine.

- Format

  flag := ibcmd(cmdstr);

- Parameter

  ✧ Entry

  cmdstr – GPIB bus command string sent to GPIB bus. There are two group commands, one is bus command sending character to the bus, the other is action command which require some actions from GPIB controller.

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error

  ✧ Bus commands are listed as follows:

Mnemonic	Hex.	Mode	Description
GTL	01	Address	Goto Local. Set addressed device to local mode.
SDC	04	Address	Selected Device Clear. Reset addressed device.
PPC	05	Address	Parallel Poll Configure.
GET	08	Address	Group Execute Trigger. Trigger all listeners on the bus.
TCT	09	Address	Take Control. Transfer control to another device.
LLO	11	Global	Local Lockout to disable front panel of all devices.
DCL	14	Global	Device Clear
PPU	15	Global	Parallel Poll Unconfigure.

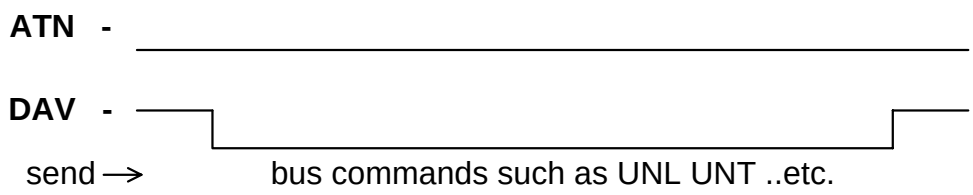
Mnemonic	Hex.	Mode	Description
SPE	18	Global	Serial Poll Enable.
SPD	19	Global	Serial Poll Disable.
REN	1F	Global	Remote Enable.
UNL	3F	Address	Unlisted. Unaddress the listener.
UNT	5F	Address	Untalk. Unaddress the talker.
MLA	35	Address	My Listen address.
MTA	55	Address	My talk address.
PPD	70	Global	Parallel poll disable.

REMARK *Global means that the commands are sent to all devices on the bus. Address means that the commands are sent to address device only.*

✧ Action commands are listed as follows:

Mnemonic	Description
LISTEN	Values following this word are listener addresses.
TALK	Value following this word is talker address.
SEC	Values following this word are secondary commands
PPE	Values following this word are parallel poll configure.
DATA	Set ATN false for data mode.
CMD	Set ATN true for command mode.
IFC	Send IFC pulse for 100ms.
EOI	Set EOI true after the last byte of output string is sent.

- GPIB bus operation : Low – true, High – false



- Example

:

Operation :

Send bus commands to trigger hp3478, addressed 23.

```
Var
      flag      :     integer;
      cmdstr    :     lstring;

cmdstr = ' UNT UNL MTA LISTEN 23 GET ';
flag := ibcmd (cmdstr, strlen (cmdstr));
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(14) Function ibspl : Serial Polling

- Introduction

  The Serial polling routine allow system controller to know whether the specified device request service by checking the returned variable srqchk low byte.

- Format

  srqchk := ibspl (gpibaddr);

- Parameter

  ✧ Entry

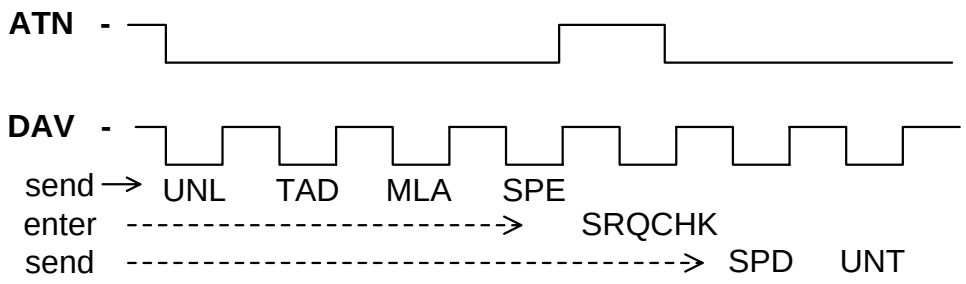
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

  srqchk – Return DIO bus data to SRQCHK integer after serial polling. Only low byte makes sense.

  BIT 7 6 5 4 3 2 1 0
  |
  SRQ
  0 : Active
  1 : None
  Bit 7 : reserved
  Bit 1 through 5 are defined depending on device.

- GPIB bus operation : Low – true, High – false



- Example

:

Operation :

Serial poll hp3478, addressed 23 after triggered to determine whether the data is ready to be read by checking return variable SRQCHK bit 0.

```
Var
      gpibaddr,   srqchk   :   integer;

gpibaddr := 23;                { HP3478 addressed 23 }
srqchk := ibspl (gpibaddr);
writeln (' serial polling status=' , srqchk);
```

(15) Function ibsts : Status of GPIB bus

- Introduction

  This routine allows you to monitor the status of GPIB bus, NEC7210 controller and etc. by checking the returned variable STATUS.

- Format

  status := ibsts (cfg);

- Parameter

  ✧ Entry

  cfg – A 16 bit configure variable to determine what status to be returned.

0 – 7	:	Return NEC7210 read register 0 – 7.
8	:	Return error number of last call routine.
9	:	Return length of output or input string.
10	:	Return Timeout interval (ms).
11	:	Return I/O port of NEC7210.
12	:	Return DMA and IRQ setting.
13 – 15	:	None

  REMARK *If bit 8 is set, then error NO. is returned as follows.*

Error No.	Description
0	No error
1	Handshake timeout
2	Interface error
3	Non-system controller call ibifc
4	Parameter setting error

  ✧ Return

  status – Returned status variable depending on CFG variable setting.

- GPIB bus operation :

  None.

- Example

  :

  Operation :

  Read error number of last call routine.

```
Var
      cfg,   status   :   integer;

cfg := $10;                    { return error NO. of last }
status := ibsts (cfg);
writeln (' status = ', status);
```

(16) Function ibtout : Setup Timeout interval

- Introduction

  This routine allows you to set timeout interval. If there is any malfunction in handshake, then the interface line is stuck until timeout interval is up.

- Format

  flag := ibtout (intvl);

- Parameter

  ✧ Entry

  intvl – Timeout period (1 – 65535).
  0 : Disable timeout routine.
  N : N units N = 1 to 32767
  N : N + 65535 units N = -1 to -32767

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation

  None

- Example

```
   :
Operation :
Set timeout interval for gpib interface bus.

Var
intvl;  flag  :  integer;

intvl := 100;                { timeout period 100 units }
flag := ibtout (intvl);
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(17) Function ibtrg : Trigger device

- Introduction

  This routine is used to trigger specified device or all devices on the interface bus by sending Group Execute Trigger (GTE) to the interface bus. If GPIBADDR is legal, the addressed device will be triggered, otherwise all devices will be triggered.

- Format

  flag := ibtrg (gpibaddr);

- Parameter

  ✧ Entry

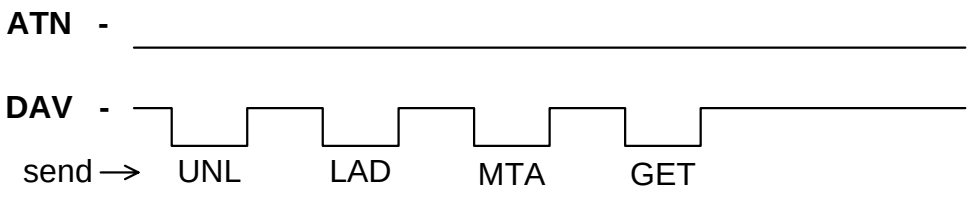
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

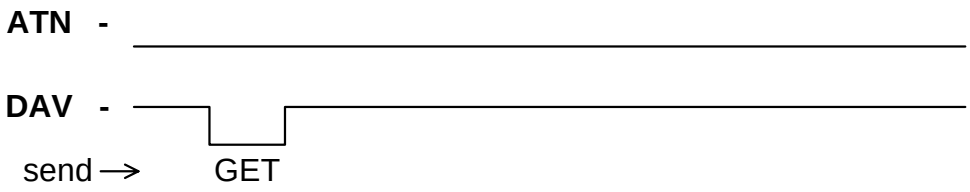
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



  ✧ Illegal GPIB address



- Example

```
    :
Operation :
Trigger hp3478, addressed 23.

Var
      gpibaddr,   flag   :   integer;

gpibaddr := 23;              { gpib address := 23 }
flag := ibtrg (gpibaddr);
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(18) Function ibrdm : Read data string from device to Memory

- Introduction

  This routine enables you to read a long data string up to 64K bytes from a device or interface bus into memory buffer with specified segment. Data string will be always placed from offset 0.

  The following condition will terminate the reading :
  - Reading number is reached.
  - Terminator or EOI is encountered.
  - Timeout.

- Format

  flag := ibrdm (gpibaddr, mseg, len);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  mseg – Segment of specified memory buffer. Be sure that the segment is not in use by your program.
  The offset start from 0 to 65535.

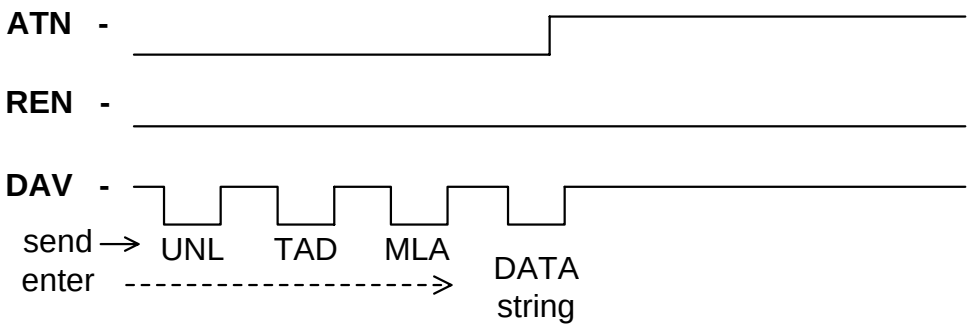
  len – Length of string 0 through 65535.

  ✧ Return

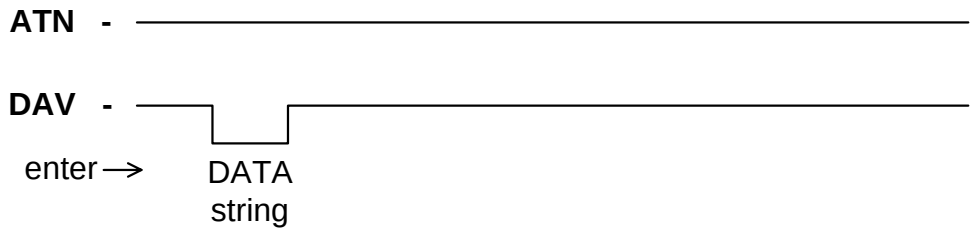
  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



✧ Illegal GPIB address



- Example

:

Operation :

Read lots of bytes of string from Hp3478, addressed 23.

```
Var
      rdno, I   :   integer;
char far            *rdbuf;
But   :   array[1..1024] of byte absolute $6000:0000;

gpibaddr := 23;                          { gpib address := 23 }
mseg := $6000;                           { memory segment := $6000 }
len := 1024;                             { collect 1024 bytes data }
flag := ibrdm (gpibaddr, mseg, len);     { collect data now }
cfg := 9;                                { Set configure := 9 for reading number }
rdno := 0;                               { Reset reading number }
rdno := ibsts (cfg);
for i := 0; to rdno do
   writeln ('cnt', i, 'data', buf[i]);
```

(19) Function ibwrtm : Write data string from Memory to device or interface

- Introduction

  This routine enables you to write a long data string up to 64K bytes from memory buffer with specified segment to a device or interface bus. The starting offset of the data number is reached.

  The following condition will terminate the writing :
  - Writing number is reached.
  - Timeout.

- Format

  flag := ibwrtm (gpibaddr, mseg, len);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

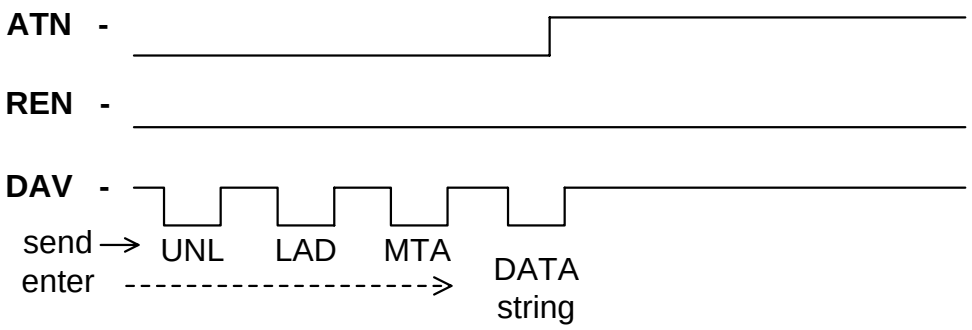
  mseg – Segment of specified memory buffer. Be sure that the segment is not in use by your program.
  The offset starts from 0 to 65535.

  len – Length of string 0 through 65535.

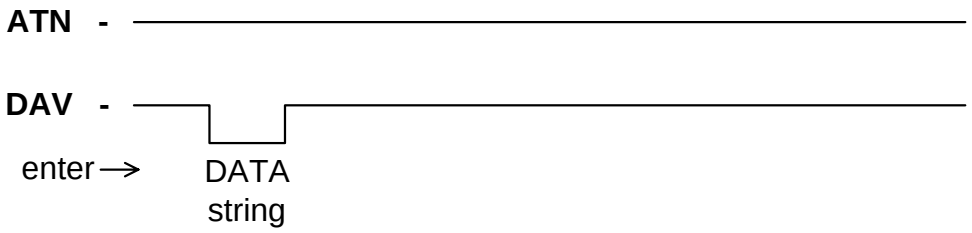
  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error

- GPIB bus operation : Low – true, High – false
  - ✧ Legal GPIB address



  - ✧ Illegal GPIB address



- Example

:

Operation :

Write a 1024 bytes of string allocated in memory with segment $6000 to Hp3478, addressed 23.

```
gpibaddr := 23;                          { gpib address := 23 }
mseg := $6000;                           { memory segment := $6000 }
len := 1024;                             { write 1024 bytes data }
flag := ibwrtm(gpibaddr, mseg, len);     { Write data now }
if (flag <> 0) then
   writeln (' call function failed, flag =' ,flag);
```

(20) Function ibxchg : Replace DOS device with GPIB device

- Introduction

  This routine enables you to replace DOS device with GPIB device. Then, you can access the GPIB device using DOS commands. The DOS devices are LPT1:, LPT2:, LPT3:, COM1: and COM2:.

- Format

  flag := ibxchg (gpibaddr, unit);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  unit – Define which device is to be replaced with GPIB device.
  1 – LPT1:
  2 – LPT2:
  3 – LPT3:
  4 – COM1:
  5 – COM2:

  ✧ Return

  flag – 0 – No error.
  1 – Handshake timeout.
  2 – Interface error.
  3 – Non-system controller call IBIFC.
  4 – Parameter setting error.

- GPIB bus operation

  None

- Example

```
    :
Operation :
Install GPIB printer, addressed 8 in place of DOS LPT1:

gpibaddr := 8;                    { gpib address := 8 }
unit := 1;                        { LPT1: }
flag := ibxchg (gpibaddr, unit);  { Change now }
Var
      gpibaddr,  unit,  flag  :  integer;
```

Example Program

- **DMM.PAS :** *Voltage Measurement with DMM (HP3478)*
- **SENDCMD.PAS :** *Voltage Measurement with DMM (HP3478) by sending commend string to interface bus.*
- **AD500.PAS :** ***R****esistance Measurement with DMM (HP3478) and AD500 programmable measurement unit.*

Programming with AX5488 Firmware Routine in Windows

Introduction

In Basic and Basic compiler language, user can call desired routine directly by defining segment and offset. But in other language, an interface function for calling the firmware routine is needed. Here, the interface function descriptions are given.

Link Interface Function

- Application Make File

input.c = application program name
gpibw.dll = the dynamic linking library
gpibw.h = the head file

```
##### Module Macro #####
APPNAME     =   input
APPSRCS     =   $ (APPNAME).c
APPOBJS     =
DLLNAME     =   gpibw
DLLSRCS     =   $ (DLLNAME).c
DLLOBJS     =   libentry.obj

##### C7 Macro #####
C7          =   1

##### Lirary Macro #####
APPLIBS     =   libw slibcew gpibw
APPMOD      =   -As
DLLLIBS     =   libw sdllcew
DLLMOD      =   -Asw
```

```
##### Include Macro #####
APPINCLS      =    $(APPNAME).h gpibw.h
DLLINCLS      =    $(APPNAME).h

##### Resource Macro #####
APPRCFILES    =    $(APPNAME).rc
DLLRCFILES    =

##### DEBUG Defined #####
DEBUG         =    1

##### Build Option Macro #####
! if $ (DEBUG)
DDEF          =    -DDEBUG
CLOPT         =    -zid –Od
MOPT          =    -zi
LOPT          =    /CO    /LI    /MAP
! else
DDEF          =
CLOPT         =    -Os
LOPT          =
! end if

##### General Macro #####
DEF           =

##### Tool Macro #####
ASM           =    masm –Mx $(MOPT) $(DDEF) $(DEF)
CC            =    cl –nologo –c $(APPMOD) –G2sw –zp –W3
                   $ (CLOPT) $(DDEF) $(DEF)
CC2           =    cl –nologo –c $(DLLMOD) –G2sw –zp –W3 –Oas
                   $ (CLPOT) $ (DDEF) $(DEF)
LINK          =    link    /NOD    /NOE $(LOPT)
RC            =    rc $(DDEF) $(DEF)
HC            =    hc
```

```
##### Interface Rules #####
.c.obj:
	$(cc)	$*.c
.asm.obj:
	$(ASM)	$*.asm;
.rc.res
	$(RC)	-r	$*.rc

##### Main (default) Target #####
goal:			$(DLLNAME).lib $(APPNAME).exe

##### Dependent for Goal and Command Line #####
$(APPNAME).exe:	$(APPSRCS:.c=.obj)	$(APPNAME).def
$(APPNAME).res
	$(LINK)	@<<
	$(APPSRCS:.c=.obj) $(APPOBJS),
	$(APPNAME).exe,
	$(APPNAME).map,
	$(APPLIBS),
	$(APPNAME).def
<<
	$(RC)	-T $(APPNAME).res
! if $ (DEBUG)
! if !$(C7)
	cvpack –p $(APPNAME).exe
! end if
	mapsym $(APPNAME).map
! end if
$(DLLNAME).obj:	$(DLLNAME).c
	$(CC2)	$*.C
$(DLLNAME).dll: $(DLLSRCS:.c = .obj)	$(DLLNAME).def
	$(LINK)	@<<
	$(DLLSRCS:.c=.obj)	$(DLLOBJS),
	$(DLLNAME).dll,
	$(DLLNAME).map,
	$(DLLLIBS),
```

```
        $(DLLNAME).def
<<
        $(RC) -T $(DLLNAME).dll
! if    $(DEBUG)
! if    !$(C7)
        cvpack    -p $(DLLNAME).dll
! end if
        mapsym $(DLLNAME).map
! end if
$(DLLNAME) .lib : $(DLLNAME).dll
        implib $(DLLNAME) .lib $(DLLNAME).def

##### Dependents #####
$(APPSRCS:.c=.obj) : $(APPINCLS)
$(APPNAME).res : $(APPINCLS)    $(APPRCFILES)
$(DLLSRCS:.c=obj) : $(DLLINCLS)
```

Summary of Interface Functions

Mnemonic	Description
ibinit	Initialize the AX5488 driver function routine
ibwrt	Write data string to device
ibrd	Read data string from device
ibifc	Interface clear to reset GPIB bus
ibeol	Establish an End of Line string for I/O of a device
ibdclr	Clear specified Device
ibllo	Local Lockout to disable device front panel operation
iblocal	Make device Go To local
ibppl	Parallel polling
ibpplc	Parallel polling configure
ibpplu	Parallel polling unconfigure
ibren	Set device to remote mode by setting Remote Enable line
ibcmd	Send command string to device
ibspl	Serial polling and return device's status
ibsts	Status of the GPIB bus
ibtout	Setup timeout interval
ibtrg	Trigger specified device

Interface Functions

(1) Function ibinit : Initialization

- Introduction

  1. Setup base address.
  2. Setup parameter to make GPIB controller work properly.
  3. Setup interrupt level and DMA channel.
  4. Automatically check firmware segment.

  This function needed to be executed only once, must be performed before any other functions.

- Format

  Segment = ibinit (ioport, gpibaddr, config);

- Parameter

  - Entry

    ioport – I/O port of GPIB controller NEC7210, default 2E1 hex..

    gpibaddr – GPIB address of device, 0 through 30, default 21.

    config – Configure of the firmware routine, depending on setting bit of 16, C15…C0.
    C5: : 1 Non-system controller
    : 0 System controller

  - Return

    Segment – Address of firmware segment. Firmware does not exist if segment = 0.

- GPIB bus operation

  None.

- Example

:

Set AX5488 operation mode as follows:
System controller, GPIB address 21, I/O port hex. 2E1.

```
int segment, gpibaddr, config, ioport;

gpibaddr = 21;              /* gpib address = 21 */
ioport = 0x2E1;             /* I/O port = 2E1 hex. */
config = 0x0000;            /* system controller */
segment = ibinit(iport, gpibaddr, config);
if (segment==0)   {
   /* AX5488 not found */
}
else {
   /* Firmware segment = segment */
}
```

(2) Function ibwrt : Write data string to device

- Introduction

  This routine writes a string followed a terminator to the specified device.

- Format

  flag = ibwrt (gpibaddr, wrtstr, wrtstr_len);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  wrtstr – Output data string.

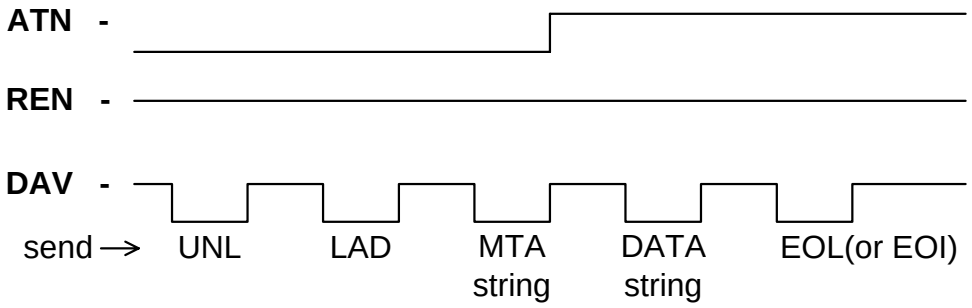
  wrtstr_len – Length of wrtstr string.

  ✧ Return

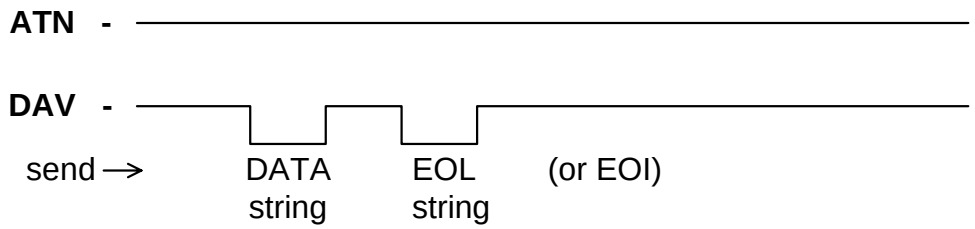
  flag –
  0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation : Low – true, High -- false

  ✧ Legal GPIB address



✧ Illegal GPIB address



- Example

:

Operation :

Write a string to Hp3478, address 23.

```
LPSTR       wrtstr;
int         gpibaddr, flag;

gpibaddr = 23;              /* GPIB address = 23 */
wrtstr = "F1T3RAN3";        /* Set up the HP3478 */
flag = ibwrt(gpibaddr, wrtstr, lstrlen(wrtstr));
if (flag!=0)   {
   /* call function failed */
}
```

(3) Function ibrd : Read data string from device

- Introduction

  This routine reads a string followed a terminator to the specified device.

- Format

  flag = ibrd (gpibaddr, rdstr, rdstr_len);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  rdstr – Input data string.

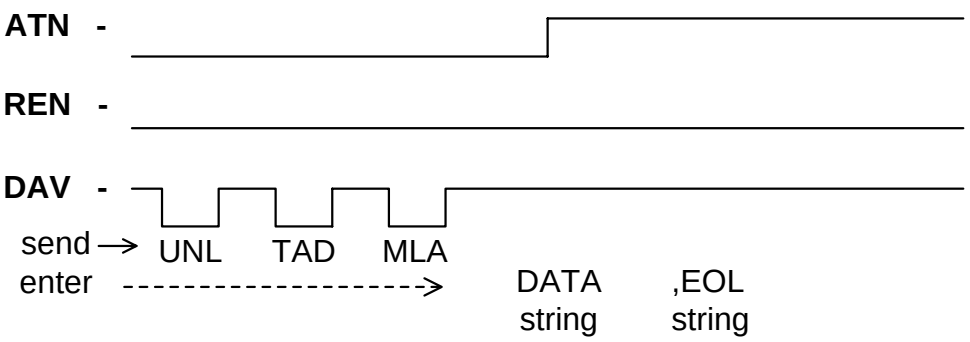
  rdstr_len – Length of rdstr string.

  ✧ Return

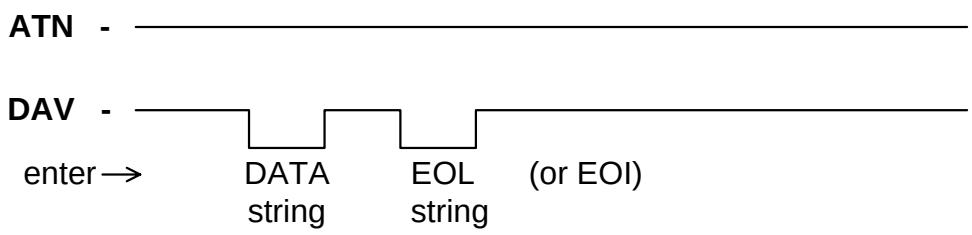
  flag –
  0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation : Low -- true, High -- false

  ✧ Legal GPIB address



✧ Illegal GPIB address



- Example

:

Operation :

Read a string from Hp3478, address 23.

```
int         flag;
char        *rdstr;

gpibaddr = 23;                  /* gpib address = 23 */
rdstr = "                   ";
flag = ibrd (gpibaddr, (LPSTR) rdstr, lstrlen((LPSTR) rdstr));
if (flag! = 0)    {
    /* call function failed */
}
```

(4) Function ibifc : Interface clear to clear GPIB bus

- Introduction

  This function, sending a IFC signal to clear GPIB interface bus, can be called by system controller only.

- Format

  flag = ibifc();

- Parameter

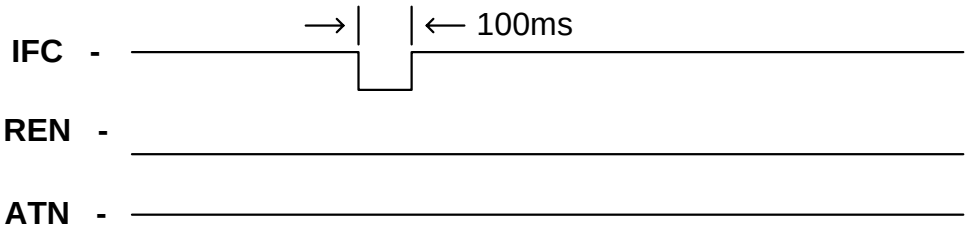
  ✧ Entry

  None

  ✧ Return

  flag – 0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation : Low –true, High – false



- Example

```
    :
Operation :
Clear all device

int         flag;

flag = ibifc();
if (flag! = 0)   {
   /* call function failed */
}
```

(5) Function ibeol : Establish an End Of Line string for I/O of a device

- Introduction

  Use this routine to establish an end of line/terminator of output and input strings for specified device, otherwise the default terminators are set.
  Default Output terminator is CR (13), LF (10).
  Default Input terminator is LF (10).

- Format

  flag = ibeol (gpibaddr, omode, oeol, oeol_len, imode, ieol);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  omode – Output terminator mode (0-2), default 0.
  0 : Terminated when encounter OEOL$ or EOI is active.
  1 : Terminated when EOI is active.
  2 : Terminated when encounter OEOL$.

  oeol – Output End Of Line string which follows output string can be set up to 8 characters, default : CR (13) and LF (10).

  oeol_len – Length of oeol string.

  imode – Input terminator mode (0-1), default 0.
  0 : Terminated when encounter IEOL% or EOI is active or input string full.
  1 : Terminated when EOI is true or input string full.

  ieol – Input End Of Line character, default is LF (10).

✧ Return

flag – 0 : No error.
-1 : Handshake timeout.
-2 : Interface error.
-3 : Non-system controller call IBIFC.
-4 : Parameter setting error.
-5 : Command out error.
-6 : Address change error.
-7 : Initialize error.
-8 : DI error.
-9 : DO error.

- GPIB bus operation

  NONE

- Example

  :

  Operation :

  Set hp3478 terminator, addressed 23.

```
int omode, imode, gpibaddr, flag, ieol;
char oeol[3];
gpibaddr = 23;                  /* gpib address = 23 */
omode = 0;
oeol[0] = 13;
oeol[1] = 10;
oeol[2] = 0;
imode = 0;
ieol = 10;
flag = ibeol (gpibaddr, omode, (LPSTR) oeol, lstrlen((LPSTR) oeol), imode, ieol);
if (flag!=0)   {
   /* call function failed */
}
```

(6) Function ibdclr : Clear specified device

- Introduction

  Send SDC (Selected Device Clear) command to clear specified device or send DCL (Device Clear) command to clear all devices.

- Format

  flag = ibdclr (gpibaddr);

- Parameter

  ✧ Entry

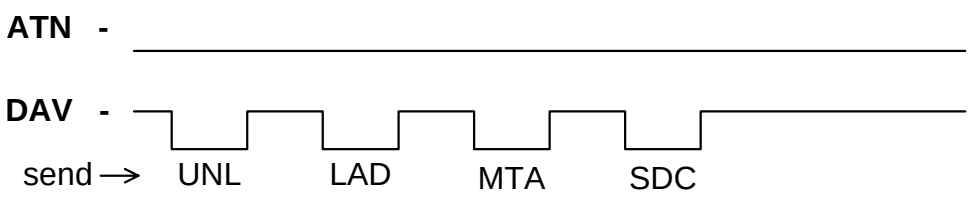
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

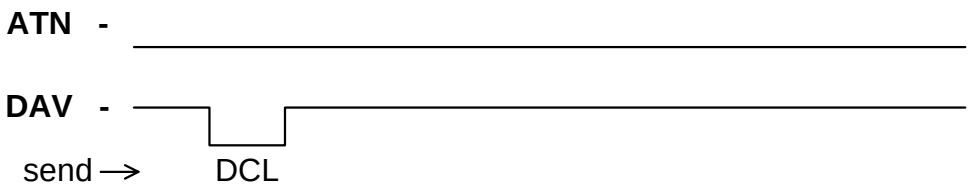
  flag – 0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address

  ✧ Illegal GPIB address

- Example

```
    :
Operation :
Clear hp3478, addressed 23.

int         flag, gpibaddr;

gpibaddr = 23;              /* gpib address = 23 */
flag = ibdclr (gpibaddr);
if (flag!=0)   {
   /* call function failed */
}
```

(7) Function ibllo : Local Lock out to disable front panel of all devices

- Introduction

  Send LLO (Local Lockout) command to disable front panel of all devices.

- Format

  flag = ibllo();

- Parameter

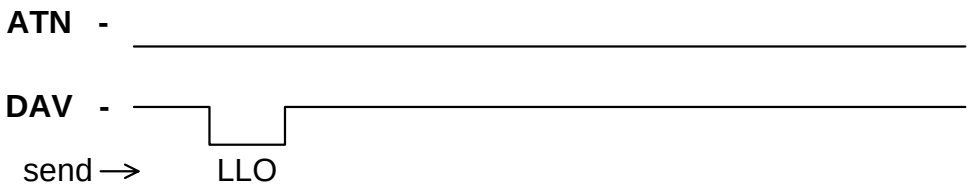
  ✧ Entry

  None

  ✧ Return

  flag – 0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation : Low – true, High – false



- Example

```
    :
Operation :
Disable front panel of all devices

int         flag;

flag = ibllo();
if (flag!=0)   {
   /* call function failed */
}
```

(8) Function iblocal : Make device Go To local

- Introduction

  Sent Go To Local (GTL) command or set Remote Enable (REN) false to enable front panel of specified device.

- Format

  flag = iblocal(gpibaddr);

- Parameter

  - Entry

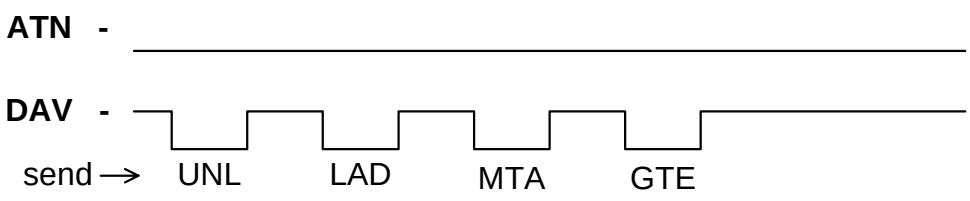
    gpibaddr – GPIB address of device, 0 through 30.

  - Return

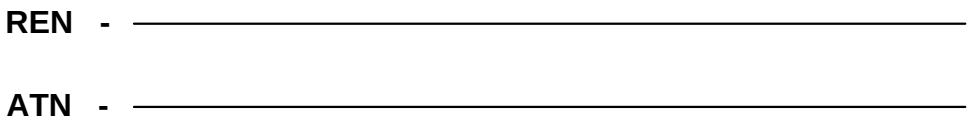
    flag – 0 : No error.
    -1 : Handshake timeout.
    -2 : Interface error.
    -3 : Non-system controller call IBIFC.
    -4 : Parameter setting error.
    -5 : Command out error.
    -6 : Address change error.
    -7 : Initialize error.
    -8 : DI error.
    -9 : DO error.

- GPIB bus operation : Low – true, High – false

  - Legal GPIB address

  - Illegal GPIB address

- Example

:

Operation :

Enable hp3478's front panel.

```
int    gpibaddr,      flag;

gpibaddr = 23;              /* gpib address = 23 */
flag = iblocal (gpibaddr);
if (flag!=0)   {
   /* call function failed */
}
```

(9) Function ibppl : Parallel Polling

- Introduction

  The Parallel polling routine allow system controller to check up to 8 devices simultaneously with SRQCHK. If any device requests service the associated bit of DIO bus is setup. After calling this routine, the DIO bus data is then transferred to variable srqchk.

  Refer to next routine ibpplc for DIO bit definition.

- Format

  srqchk = ibppl();

- Parameter

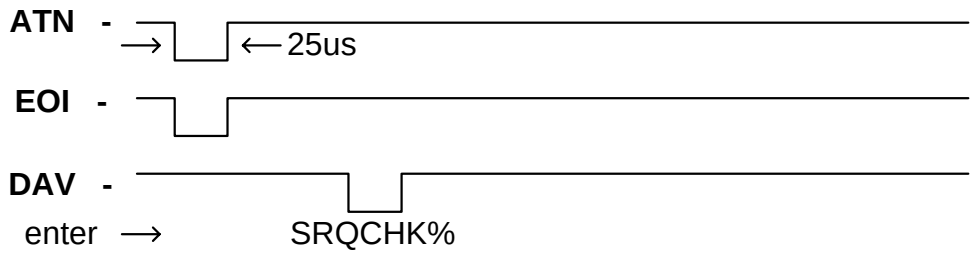
  ✧ Entry

  None

  ✧ Return

  srqchk – Return DIO bus data to SRQCHK integer after parallel polling. Only low byte makes sense.

- GPIB bus operation : Low – true, High – false



- Example

```
   :
int   srqchk;

srqchk = ibppl();
/* parallel polling status = srqchk */
```

(10) Function ibpplc : Parallel Polling Configure

- Introduction

  The Parallel polling configure routine allows device to specify a DIO bus line to respond to parallel poll. It uses integer CONFIG bit 0 through 2 to select DIO bus line, bit 3 to determine the response state of the corresponding line.

- Format

  flag = ibpplc(gpibaddr, config);

- Parameter

  ✧ Entry

  gpibaddr – GPIB address of device, 0 through 30.

  config – An integer to configure a device to respond to parallel polling. Only the low nibble makes sense.

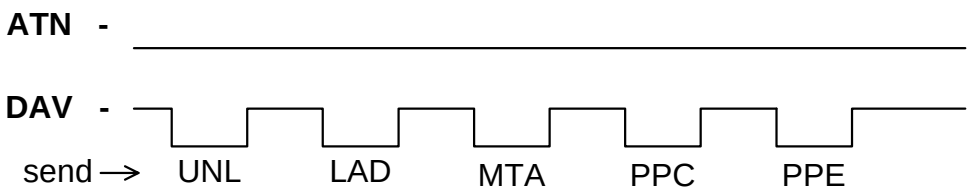
BIT	2	1	0
	0	0	0 – DIO1
	0	0	1 – DIO2
		:	
	1	1	1 – DIO8

  BIT 3 – Determine the state of data line(DIO#).
  0 : low state
  1 : high state

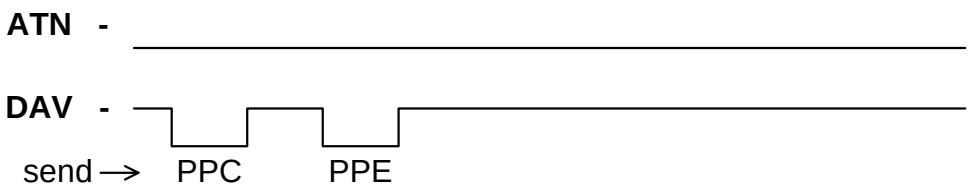
  ✧ Return

  flag –
  0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation : Low – true, High – false
  - ✧ Legal GPIB address



  - ✧ Illegal GPIB address



- Example

:

Operation :

Configure hp3478, addressed 23.

```
int config,    gpibaddr,    flag;

config = 0x00;                  /* select DIO1, reflect with low state */
gpibaddr = 23;                  /* gpib address = 23 */
flag = ibpplc (gpibaddr, config);
if (flag!=0)    {
    /* call function failed */
}
```

(11) Function ibpplu : Parallel Polling Unconfigure

- Introduction

  The Parallel polling Unconfigure routine is used to tell specified device or all devices on the interface bus not to respond to the parallel polling. If GPIBADDR is legal, the addressed device will be unconfigured, otherwise all devices will be unconfigured.

- Format

  flag = ibpplu(gpibaddr);

- Parameter

  ✧ Entry

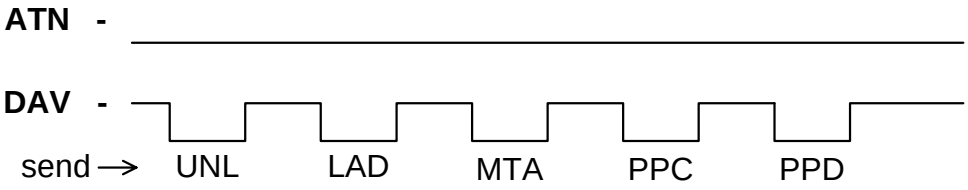
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

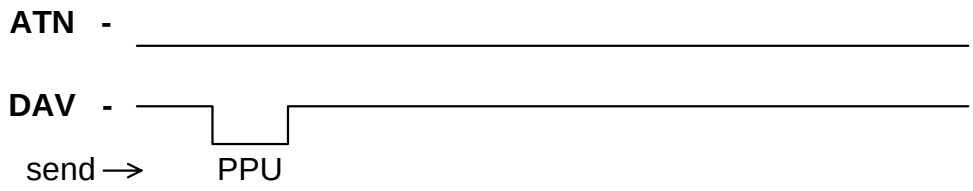
  flag –
  0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



✧ Illegal GPIB address



- Example

:

Operation :

Unconfigure hp3478, addressed 23.

```
int    gpibaddr,    flag;

gpibaddr = 23;                /* gpib address = 23 */
flag = ibpplu (gpibaddr);
if (flag!=0)    {
    /* call function failed */
}
```

(12) Function ibren : Remote Enable

- Introduction

  This routine is used to set specified device or all devices on the interface bus to remote mode. If GPIBADDR is legal, the addressed device will be in remote mode, otherwise all devices will be in remote mode.

- Format

  flag = ibren(gpibaddr);

- Parameter

  ✧ Entry

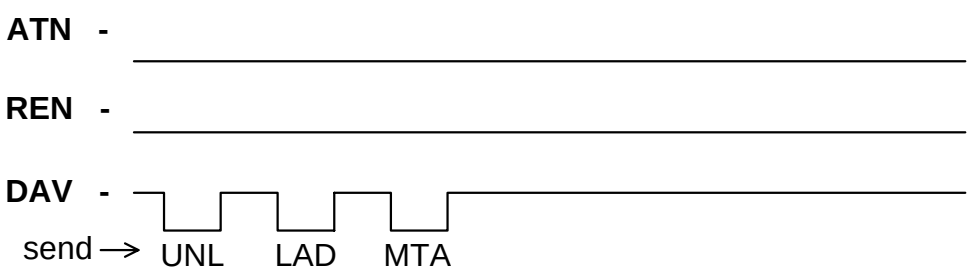
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

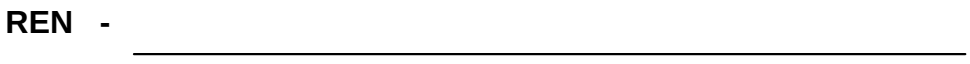
  flag –
  0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address

  ✧ Illegal GPIB address

- Example

:

Operation :

Set hp3478, addressed 23 to remote mode.

```
gpibaddr = 23;                /* gpib address = 23 */
flag = ibren (gpibaddr);
if (flag!=0)    {
   /* call function failed */
}
```

(13) Function ibcmd : Send command code

- Introduction

  To send command to GPIB interface bus, or to control some devices simultaneously, use this routine.

- Format

  flag = ibcmd(cmdstr, cmdstr_len);

- Parameter

  ✧ Entry

  cmdstr – Command string sent to GPIB bus. There are two group of commands, one is bus command for sending character to the bus, the other is action command which require some actions from GPIB controller.

  cmdstr_len – Length of cmdstr string.

  ✧ Return

  flag –
  0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

✧ Bus commands are listed as follows:

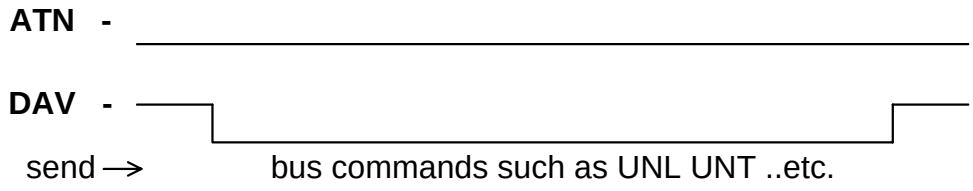
Mnemonic	Hex.	Mode	Description
GTL	01	Address	Go To Local. Set addressed device to local mode.
SDC	04	Address	Selected Device Clear. Reset addressed device.
PPC	05	Address	Parallel Polling Configure.
GET	08	Address	Group Execute Trigger. Trigger all listeners on the bus.
TCT	09	Address	Take Control. Transfer control to another device.
LLO	11	Global	Local Lockout to disable front panel of all device.
DCL	14	Global	Device Clear
PPU	15	Global	Parallel Poll Unconfigure.
SPE	18	Global	Serial Poll Enable.
SPD	19	Global	Serial Poll Disable.
REN	1F	Global	Remote Enable.
UNL	3F	Address	Unlisted. Unaddress the listener.
UNT	5F	Address	Untalk. Unaddress the talker.
MLA	35	Address	My Listen address.
MTA	55	Address	My talk address.
PPD	70	Global	Parallel poll disable.

REMARK *Global means that the commands are sent to all devices on the bus. Address means that the commands are sent to address device only.*

✧ Action commands are listed as follows:

Mnemonic	Description
LISTEN	Values following this word are listener addresses.
TALK	Value following this word is talker address.
SEC	Values following this word are secondary commands
PPE	Values following this word are parallel poll configure.
DATA	Set ATN false for data mode.
CMD	Set ATN true for command mode.
IFC	Send IFC pulse for 100ms.
EOI	Set EOI true after the last byte of output string is sent.

- GPIB bus operation : Low – true, High – false



- Example

:

Operation :

Send bus commands to trigger hp3478, addressed 23.

```
int             flag;
LPSTR           *cmdstr;

cmdstr = "UNT UNL MTA LISTEN 23 GET";
flag = ibcmd (cmdstr, lstrlen (cmdstr));
if (flag!=0)    {
   /* call function failed */
}
```

(14) Function ibspl : Serial Polling

- Introduction

  The Serial polling routine allows system controller to know whether the specified device is requesting service or not by checking the low byte of the returned variable, srqchk.

- Format

  srqchk = ibspl (gpibaddr);

- Parameter

  ✧ Entry

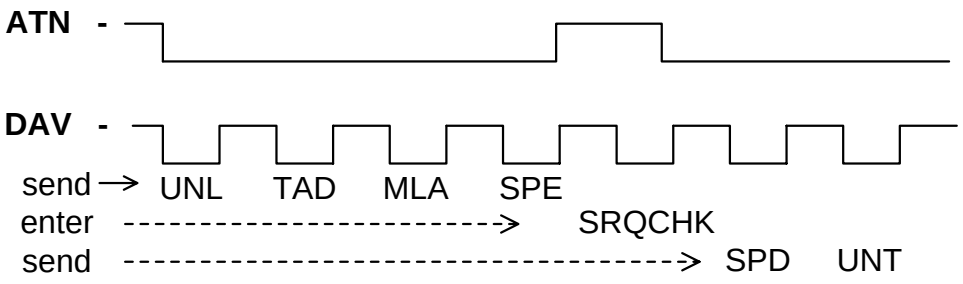
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

  srqchk – Return DIO bus data to SRQCHK integer after serial polling. Only low byte makes sense.

  BIT 7 6 5 4 3 2 1 0
  |
  SRQ
  0 : Active
  1 : None
  Bit 7 : reserved
  Bit 0 through 6 are defined depending on device.

- GPIB bus operation : Low – true, High – false



- Example

:

Operation :

Serial poll hp3478, addressed 23 after triggered to determine whether the data is ready to be read by checking return variable SRQCHK bit 0.

```
int    gpibaddr,    srqchk;

gpibaddr = 23;                  /* HP3478 addressed 23 */
srqchk = ibspl (gpibaddr);
/* serial polling status = srqchk */
```

(15) Function ibsts : Status of GPIB bus

- Introduction

  This routine allows you to monitor the status of GPIB bus, NEC7210 controller and etc. by checking the returned variable STATUS.

- Format

  status = ibsts (cfg);

- Parameter

  ✧ Entry

  cfg – A 16 bit configure variable specified which status is returned.

0 – 7	:	Return NEC7210 read register 0 – 7.
8	:	Return error number of last call routine.
9	:	Return length of output or input string.
10	:	Return Timeout interval.
11	:	Return I/O port of NEC7210.
12	:	Return DMA and IRQ setting.
13 – 15	:	None

REMARK *If bit 8 is set, then error NO. type returned is as follows.*

Error No.	Description
0	No error.
-1	Handshake timeout.
-2	Interface error.
-3	Non-system controller call IBIFC.
-4	Parameter setting error.
-5	Command out error.
-6	Address change error.
-7	Initialize error.
-8	DI error.
-9	DO error.

✧ Return

status – Returned status variable depending on CFG variable setting.

- GPIB bus operation :

  None.

- Example

  :

  Operation :

  Read error number of last call routine.

```
int   cfg, status;

cfg = 0x10;          /* return error NO. of last call routine */
status = ibsts (cfg);
/* Return status */
```

(16) Function ibtout : Setup Timeout interval

- Introduction

  This routine allows you to set timeout interval. If there is any malfunction in handshake, then the interface line is stuck until timeout interval is up.

- Format

  flag = ibtout (intvl);

- Parameter

  ✧ Entry

  intvl – Timeout period (1 – 65535).
  0 : Disable timeout routine.
  N : N units N = 1 to 32767
  N : N + 65535 units N = -1 to -32767

  ✧ Return

  flag –
  0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation

  None

- Example

```
    :
Operation :
Set timeout interval for gpib interface bus.

int   intvl,   flag;

intvl = 100;              /* timeout period 100 units */
flag = ibtout (intvl);
if (flag! = 0)   {
   /* call function failed */
}
```

(17) Function ibtrg : Trigger device

- Introduction

  This routine is used to trigger specified device or all devices on the interface bus by sending Group Execute Trigger (GTE) to the interface bus. If GPIBADDR is legal, the addressed device will be triggered, otherwise all devices will be triggered.

- Format

  flag = ibtrg (gpibaddr);

- Parameter

  ✧ Entry

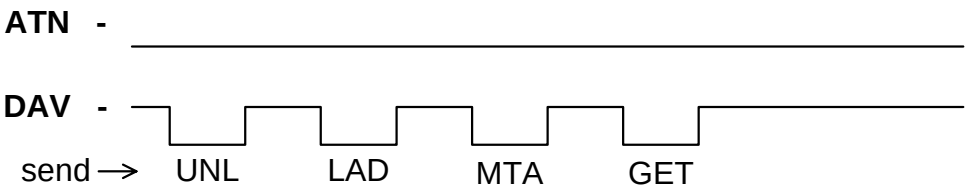
  gpibaddr – GPIB address of device, 0 through 30.

  ✧ Return

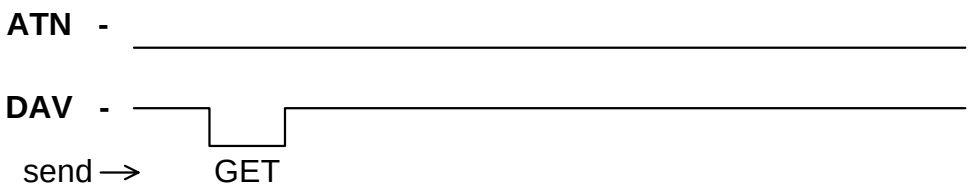
  flag –
  0 : No error.
  -1 : Handshake timeout.
  -2 : Interface error.
  -3 : Non-system controller call IBIFC.
  -4 : Parameter setting error.
  -5 : Command out error.
  -6 : Address change error.
  -7 : Initialize error.
  -8 : DI error.
  -9 : DO error.

- GPIB bus operation : Low – true, High – false

  ✧ Legal GPIB address



✧ Illegal GPIB address



- Example

:

Operation :

Trigger hp3478, addressed 23.

```
int    gpibaddr,    flag;

gpibaddr = 23;              /* gpib address = 23 */
flag = ibtrg (gpibaddr);

if (flag != 0)    {
    /* call function failed */
}
```

Introduce IEEE-488 (GPIB)

General Introduction

IEEE-488, 24 lines bus consists of 8 GNDS, 8 DATA lines, 8 CONTROL lines was designed for use in network for instruments. The maximum transmission rate is 1 mega bits per second.

The network may contain up to 15 devices, which are classified into 3 types.

- Talker : *An device transmits data on the bus. Ex. Voltmeters…*
- Listener : *An device receives data from the bus. Ex. Printer, XY-plotter…*
- Controller : *An device designates which device can talk or listen. Ex. PC…..*

In gpib system, only one device can talk at a time, but multiple devices can listen at the same time, and only one system controller is allowed.

In addition, be sure of your network of instrument meeting the following limitation.

- Limitation
  - ✧ Quantity of devices < 15
  - ✧ Sum of length of cables < 20 m or L < 2N (m)
    N : device number

IEEE-488 Connector Introduction

There are 24 pin on the IEEE-488 connector described as follows:

- Data line : 8

Mnemonic	Pin No.	Description
DIO 1	1	for transfer commands, address, data
DIO 2	2	for transfer commands, address, data
DIO 3	3	for transfer commands, address, data
DIO 4	4	for transfer commands, address, data
DIO 5	13	for transfer commands, address, data
DIO 6	14	for transfer commands, address, data
DIO 7	15	for transfer commands, address, data
DIO 8	16	for transfer commands, address, data

- Management line : 5, low true

Mnemonic	Pin No.	Description
EOI	5	The end of multiple transferred bytes or executing a parallel polling in conjunction with ATN
IFC	9	Clear the interface functions
SRQ	10	Used to request the controller for service
ATN	11	Attention : low for command mode, high for data mode
REN	17	Used to select remote or local control of the devices

- Handshake line : 3, low true

Mnemonic	Pin No.	Description
DAV	6	Data valid
NRFD	7	Not ready for data
NDAC	8	No data accepted

- GND line : 8

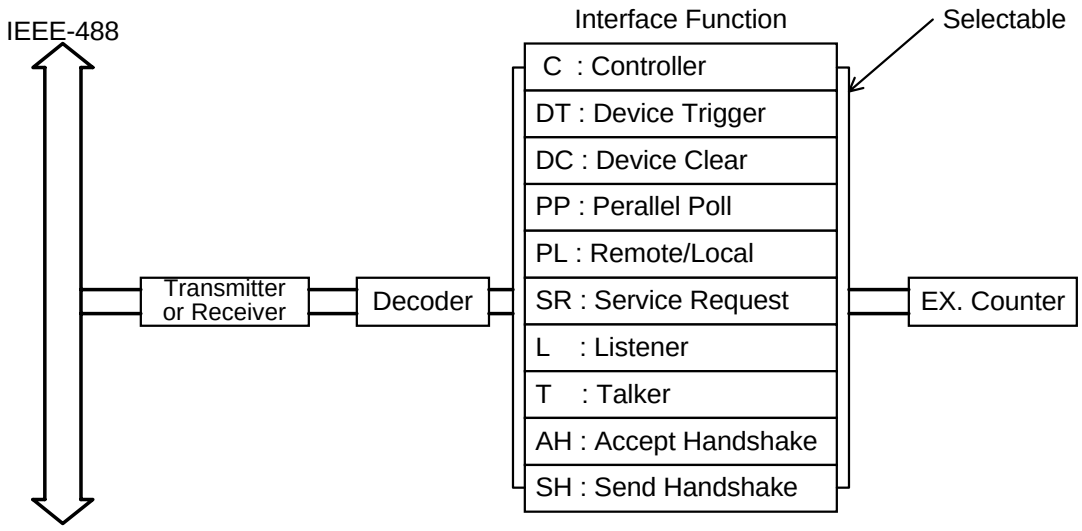
Mnemonic	Pin No.	Description
SHILD	12	Shielding Ground
GND	18	Twisted pair with DAV
GND	19	Twisted pair with NRFD
GND	20	Twisted pair with NDAC
GND	21	Twisted pair with IFC
GND	22	Twisted pair with SRQ
GND	23	Twisted pair with ATN
GND	24	Signal Ground

Interface Function

When you implement a system in gpib interface bus, you must know how to select device, what functions the device has. The functions are described as follows:

Mnemonic	Description
C	Controller
DT	Device trigger
DC	Device Clear
RP	Parallel polling
RL	Remote and Local
SR	Service Request
L	Listener
T	Talker
AH	Accepting Handshake
SH	Sending Handshake

A device may have several functions.

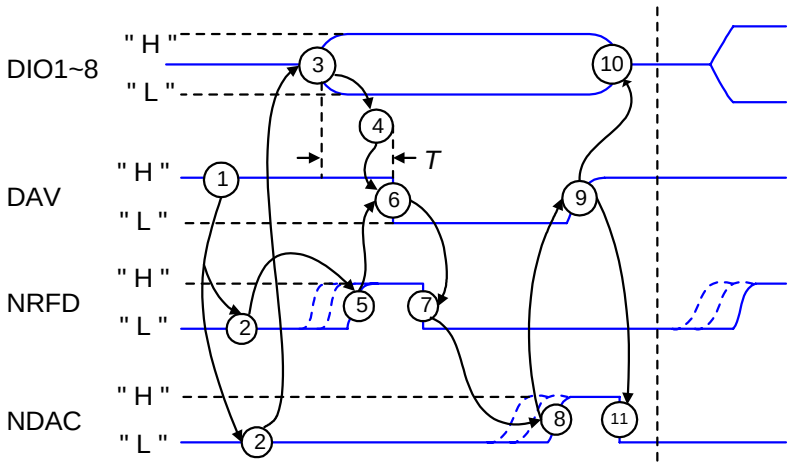


Handshake Timing Description

In gpib system, communication between devices is under control by handshake sequence.

There are one talker and several listeners in the following example:

- Initialization : as step 1, 2 in the diagram.
  - DAV = "H" is send by talker.
  - NRFD & NDAC = "L" is sent by listeners.
- Data transfer : as step 3, 4, 5, 6, 7
  - The last listener is ready for data by setting NRFD "H", as step 5.
  - The talker detect NRFD "H", then pass data to bus and latterly set DAV "L" to tell listener data valid, as step 3, 4, 6.
  - Upon detecting DAV "L", listeners set NRFD "L" and receive data from the bus, as step 7.
- Completion : as step 8, 9, 10
  - When the last device has received data, it set NADC "H", as step 8.
  - Upon detecting NADC "H", the talker set DAV "H" for transferring next data, as step 9, 10.
  - Upon detecting DAV "H", the listeners set NRFD "L" for receiving next data, as step 11.



IEEE-488 Bus Commands and Address

In command mode, system controller can pass either command or address to the bus for specified device or all devices.

Bus commands

Mnemonic	Hex.	Mode	Description
GTL	01	Address	Goto Local. Set addressed device to local mode.
SDC	04	Address	Selected Device Clear. Clear addressed device.
PPC	05	Address	Parallel Poll Configure.
GET	08	Address	Group Execute Trigger. Trigger all listeners on the bus.
TCT	09	Address	Take Control. Transfer control to another device.
LLO	11	Global	Local Lockout to disable front panel of all devices.
DCL	14	Global	Device Clear.
PPU	15	Global	Parallel Poll Unconfigure.
SPE	18	Global	Serial Poll Enable.
SPD	19	Global	Serial Poll Disable.
UNL	3F	Address	Unlisted. Unaddress the listener
UNT	5F	Address	Untalk. Unaddress the talker.

REMARK *Global means that the commands are sent to all devices on the bus. Address means that the commands are sent to address device only.*

Overview of Bus commands and address

--ASCII – IEEE 488 BUS Messages (Commands and Addresses) Hex Codes--

MSD / LSD	0		1		2		3		4		5		6		7	
	ASCII	MSG	ASCII	MSG	ASCII	MSG1	ASCII	MSG1	ASCII	MSG1	ASCII	MSG	ASCII	MSG	ASCII	MSG
0	NUL		DLE		SP	00	0	16	@	00	P	16	`	MEANING DEFINED BY PCG CODE	p	MEANING DEFINED BY PCG CODE
1	SOH	GTL	DC1	LLO	!	01	1	17	A	01	Q	17	a		q	
2	STX		DC2		"	02	2	18	B	02	R	18	b		r	
3	ETX		DC3		#	03	3	19	C	03	S	19	c		s	
4	EOT	SDC	DC4	DCL	$	04	4	20	D	04	T	20	d		t	
5	ENQ	PPC	NAK	PPU	%	05	5	21	E	05	U	21	e		u	
6	ACK		SYN		&	06	6	22	F	06	V	22	f		v	
7	BEL		ETB		'	07	7	23	G	07	W	23	g		w	
8	BS	GET	CAN	SPE	(	08	8	24	H	08	X	24	h		x	
9	HT	TCT	EM	SPD	)	09	9	25	I	09	Y	25	i		y	
A	LF		SUB		*	10	:	26	J	10	Z	26	j		z	
B	VT		ESC		+	11	;	27	K	11	[	27	k		(	
C	FF		FS		,	12	<	28	L	12	\	28	l		|	
D	CR		GS		-	13	=	29	M	13	]	29	m		)	
E	SO		RS		.	14	>	30	N	14	^	30	n		~	
F	SI		US		/	15	?	UNL	O	15	_	UNT	o		DEL	

Addressed Command Group

Universal Command Group

Listen Address Group

Talk Address Group

Secondary Command Group

Primary Command Group (PCG)

NOTE

1. Device Address messages shown in decimal

2. Message codes are:
   - **DCL** – Device Clear
   - **DGT** – Device Trigger
   - **GTL** – Go to Local
   - **LLO** – Local Lockout
   - **PPC** – Parallel Poll Cofigure
   - **PPU** – Parallel Poll Unconfigure
   - **SDC** – Serial Device Clear
   - **SPD** – Serial Poll Disable
   - **SPE** –Serial Poll Enable

3. ATN off, Bus data is ASCII; ATN on, Bus data is an IEEE MSG.

Appendix A ASCII Table

CHAR	BINARY	HEX	DEC	COMMAND	CHAR	BINARY	HEX	DEC	COMMAND
NULL	0000000	00	0		SPACE	0100000	20	32	LA0
SOH	0000001	01	1	GTL	!	0100001	21	33	LA1
STX	0000010	02	2		"	0100010	22	34	LA2
ETX	0000011	03	3		#	0100011	23	35	LA3
EOT	0000100	04	4	SDC	$	0100100	24	36	LA4
ENQ	0000101	05	5	PPC	%	0100101	25	37	LA5
ACK	0000110	06	6		&	0100110	26	38	LA6
BELL	0000111	07	7		`	0100111	27	39	LA7
BS	0001000	08	8		(	0101000	28	40	LA8
HT	0001001	09	9		)	0101001	29	41	LA9
LF	0001010	0A	10		*	0101010	2A	42	LA10
VT	0001011	0B	11		+	0101011	2B	43	LA11
FF	0001100	0C	12		'	0101100	2C	44	LA12
CR	0001101	0D	13		-	0101101	2D	45	LA13
SO	0001110	0E	14		.	0101110	2E	46	LA14
SI	0001111	0F	15		/	0101111	2F	47	LA15
DLE	0010000	10	16		0	0110000	30	48	LA16
DC1	0010001	11	17	LLO	1	0110001	31	49	LA17
DC2	0010010	12	18		2	0110010	32	50	LA18
DC3	0010011	13	19		3	0110011	33	51	LA19
DC4	0010100	14	20	DCL	4	0110100	34	52	LA20
NAK	0010101	15	21	PPU	5	0110101	35	53	LA21
SYNC	0010110	16	22		6	0110110	36	54	LA22
ETB	0010111	17	23		7	0110111	37	55	LA23
CAN	0011000	18	24	SPE	8	0111000	38	56	LA24
EM	0011001	19	25	SPD	9	0111001	39	57	LA25
SUB	0011010	1A	26		:	0111010	3A	58	LA26
ESC	0011011	1B	27		;	0111011	3B	59	LA27
FS	0011100	1C	28		<	0111100	3C	60	LA28
GS	0011101	1D	29		=	0111101	3D	61	LA29
RS	0011110	1E	30		>	0111110	3E	62	LA30
US	0011111	1F	31		?	0111111	3F	63	UNL

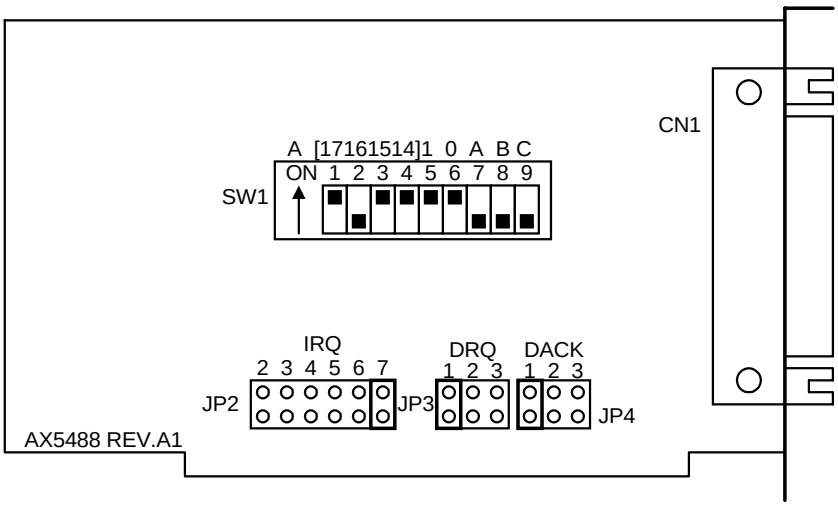
CHAR	BINARY	HEX	DEC	COMMAND	CHAR	BINARY	HEX	DEC	COMMAND
	1000000	40	64	TA0		1100000	60	96	SC0
A	1000001	41	65	TA1	a	1100001	61	97	SC1
B	1000010	42	66	TA2	b	1100010	62	98	SC2
C	1000011	43	67	TA3	c	1100011	63	99	SC3
D	1000100	44	68	TA4	d	1100100	64	100	SC4
E	1000101	45	69	TA5	e	1100101	65	101	SC5
F	1000110	46	70	TA6	f	1100110	66	102	SC6
G	1000111	47	71	TA7	g	1100111	67	103	SC7
H	1001000	48	72	TA8	h	1101000	68	104	SC8
I	1001001	49	73	TA9	i	1101001	69	105	SC9
J	1001010	4A	74	TA10	j	1101010	6A	106	SC10
K	1001011	4B	75	TA11	k	1101011	6B	107	SC11
L	1001100	4C	76	TA12	l	1101100	6C	108	SC12
M	1001101	4D	77	TA13	m	1101101	6D	109	SC13
N	1001110	4E	78	TA14	n	1101110	6E	110	SC14
O	1001111	4F	79	TA15	o	1101111	6F	111	SC15
P	1010000	50	80	TA16	p	1010000	70	112	SC16
Q	1010001	51	81	TA17	q	1110001	71	113	SC17
R	1010010	52	82	TA18	r	1110010	72	114	SC18
S	1010011	53	83	TA19	s	1110011	73	115	SC19
T	1010100	54	84	TA20	t	1110100	74	116	SC20
U	1010101	55	85	TA21	u	1110101	75	117	SC21
V	1010110	56	86	TA22	v	1110110	76	118	SC22
W	1010111	57	87	TA23	w	1110111	77	119	SC23
X	1011000	58	88	TA24	x	1111000	78	120	SC24
Y	1011001	59	89	TA25	y	1111001	79	121	SC25
Z	1011010	5A	90	TA26	z	1111010	7A	122	SC26
[	1011011	5B	91	TA27	{	1111011	7B	123	SC27
\	1011100	5C	92	TA28	|	1111100	7C	124	SC28
]	1011101	5D	93	TA29	}	1111101	7D	125	SC29
^	1011110	5E	94	TA30	-	1111110	7E	126	SC30
_	1011111	5F	95	UNT	DEL	1111111	7F	127	SC31

Appendix B NEC7210 Read/Write Registers

Read	Register	Bit Contents							
		7	6	5	4	3	2	1	0
0	Data In	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0
1	Interrupt Status 1	CPT	APT	DET	END	DEC	ERR	DO	DI
2	Interrupt Status 2	INT	SRQI	LOK	REM	CO	LOKC	REMC	ADSC
3	Serial Poll Status	S8	PEND	S6	S5	S4	S3	S2	S1
4	Address Status	CIC	ATN	SPMS	LPAS	TPAS	LA	TA	MJMN
5	Command Pass Through	CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0
6	Address 0	X	DT0	DL0	AD50	AD40	AD30	AD20	AD10
7	Address 1	EOI	DT1	DL1	AD51	AD41	AD31	AD21	AD11

Write	Register	Bit Contents							
		7	6	5	4	3	2	1	0
0	Byte Out	BO7	BO6	BO5	BO4	BO3	BO2	BO1	BO0
1	Interrupt Mask 1	CPT	APT	DET	END	DEC	ERR	DO	DI
2	Interrupt Mask 2	0	SRQI	DMAO	DMAI	CO	LOKC	REMC	ADSC
3	Serial Poll	S8	rsv	S6	S5	S4	S3	S2	Mode
4	Address Mode	ton	lon	TRM1	TRM0	0	0	ADM1	ADM0
5	Auxiliary Mode	CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0
6	Address 0/1	ARS	DT0	DL	AD5	AD4	AD3	AD2	AD1
7	End of string	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0

Appendix C Jumpers and Switch Location



Appendix D Technique Reference

Interrupt

In the firmware routine, there are three types of interrupt requests including timeout, SRQ and ERROR. The associated interrupt event defined in Basica version 3.0 & 3.3 only are function key 19, 20 and ERROR. It is necessary to prepare the interrupt service routine, if you want to handle the interrupts.

The following procedures show you how to enable interrupt.

- Enable the interrupt by setting the associated bit of CONFIG% of IBINIT% routine as follows:

Bit	Function key	Interrupt Description
9	F20	SRQ : when SRQ line is set true by some devices
10	F19	Timeout : handshake is stuck
11	ERROR	ERROR : some error happen

- Prepare interrupt service routine as follows:

```
ON KEY (20) GOSUB 100          ' SRQ service routine
ON KEY (19) GOSUB 200          ' Timeout service routine
ON ERROR  GOSUB 300            ' ERROR service routine

100     ' SRQ service routine
.
150     return
200     ' Timeout service routine
.
250     return
300     ' ERROR service routine
.
350     return
```

DMA

This firmware routine supports DMA operation, single and block transfer modes. In single mode, DMA working in background, you can run your application program regardless of the DMA operation. In block mode, DMA occupies the bus until transfers are completed.

The following procedures show you how to enable DMA.

- Setting the associated bit of CONFIG% of IBINIT% routine as follows:

Bit	Description
0	Bit 0 & 1 determine DMA channel. 00: None.
1	01: Channel 1. 10: Channel 2. 11: Channel 3.
14	0: Single transfer mode. 1: Block transfer mode.
15	0 : DMA wait until completed. 1: DMA background.

- Setup 8237 chip's registers to initialize DMA:
  - ✧ Setup Page register
  - ✧ Setup Address register
  - ✧ Setup Count register
  - ✧ Setup Mode register
- Setup 8237 chip's request registers to enable DMA:

  Refer to 8237 DMA controller chip for more detail description.